

AFIT/DS/ENG/99-05

EVOLVING COMPACT DECISION RULE SETS

DISSERTATION

Robert Evan Marmelstein
Major, USAF

AFIT/DS/ENG/99-05

DTIC QUALITY INSPECTED 1

Approved for public release; distribution unlimited

19990616 030

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

EVOLVING COMPACT DECISION RULE SETS

DISSERTATION

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Robert Evan Marmelstein, B.S.E.E., M.S.E.E.

Major, USAF

June, 1999

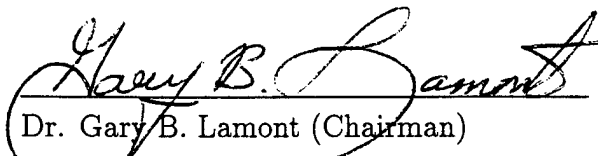
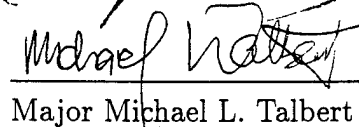
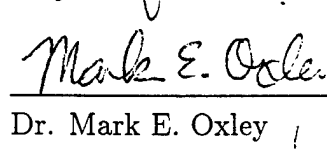
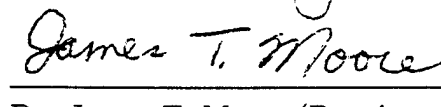
Approved for public release; distribution unlimited

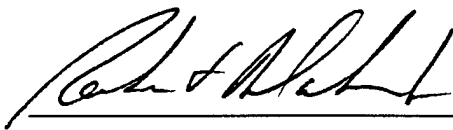
EVOLVING COMPACT DECISION RULE SETS

Robert Evan Marmelstein, B.S.E.E., M.S.E.E.

Major, USAF

Approved:

 Dr. Gary B. Lamont (Chairman)	<u>17 MAY 99</u> Date
 Major Michael L. Talbert	<u>17 May 99</u> Date
 Dr. Mark E. Oxley	<u>19 May 99</u> Date
 Dr. James T. Moore (Dean's Representative)	<u>19 May 99</u> Date


Dr. Robert A. Calico, Jr.
Dean, Graduate School of Engineering

Acknowledgements

The last three years at AFIT have been a challenging experience. I have learned a great deal here and am deeply indebted to the many people who have made the success of this research possible. To this end, I would like to take this opportunity to give credit where credit is due.

First, I would like to thank my advisor, Dr. Gary Lamont, for his insight, patience, and advice which helped me stay focused during this long process. Of special note was his encouragement to look at issues from a different perspective. To my committee members (past and present), Major Mike Talbert, Capt Sam Gardner, Dr. Mark Oxley, Dr. James Moore and Dr. Marty DeSimio, I thank you for the valuable advice and guidance you have provided. Additional thanks are extended to the following individuals: Dr. Robert Ewing for his support in obtaining funding for the follow-on GRaCCE research; Dr. Louis Tamburino and Dr. Mateen Rizki for their interest in my work; Capt Lonnie Hammack for his work on cGRaCCE; Capt Scott Brown for his review of this dissertation; and lastly, Mr. Dave Doak for his help in solving my many Hawkeye-related problems.

Most importantly, I thank my wife, Victoria, and my children Alan and Sarah. Their constant love, understanding and support gave me the mental and spiritual stamina to complete this effort.

Robert Evan Marmelstein

Table of Contents

	Page
Acknowledgements	iii
List of Figures	xi
List of Tables	xvii
Abstract	xix
 I. Introduction	 1
1.1 What is Data Mining?	1
1.2 Applicability of Data Mining to the USAF Mission . .	4
1.2.1 Augmenting the OODA Loop	4
1.2.2 Defense Against Cyber-Attack	5
1.2.3 Weather Prediction	6
1.2.4 Additional Applications	6
1.3 Research Problem	6
1.4 Approach and Design Goals	7
1.5 Assumptions	8
1.6 Summary	9
 II. Background	 10
2.1 Introduction to Decision Rule Induction	10
2.2 Decision Trees	11
2.2.1 Fundamentals of Decision Tree Induction . . .	11
2.2.2 Univariate Partition Approach	14
2.2.3 Oblique Partition Approach	16

	Page
2.2.4 Pruning the Tree Structure	20
2.2.5 Variant Methods	23
2.3 Piecewise Linear Classifiers	26
2.4 Evolutionary Methods	28
2.4.1 Genetic Decision Tree Algorithms	29
2.4.2 Classifier Systems	30
2.4.3 Genetic Program-Based Classifiers	33
2.4.4 Mining the Genetic Program	36
2.5 Association Rules	37
2.6 Induction of Bayesian Networks	38
2.7 Rule Extraction	40
2.8 Summary	43
III. Preliminary Issues	45
3.1 Effects of Dimensionality	45
3.2 Feature Selection	46
3.3 Feature Extraction	48
3.4 Processing Very Large Databases	50
3.5 Discrete Features	52
3.6 Modality of Data	54
3.7 Over-fitting and Training Set Composition	55
3.8 Missing Data	57
3.9 Normalization	58
3.10 Identifying Interesting Information	59
3.11 Summary	61

	Page
IV. Overview of the GRaCCE Algorithm	63
4.1 Introduction to GRaCCE	63
4.2 Preprocessing Phase	65
4.2.1 Feature Selection	65
4.2.2 Winnowing	66
4.3 Partition Generation Phase	67
4.3.1 Fundamentals of Partition Estimation	67
4.3.2 Global Partition Generation	70
4.3.3 Local Partition Generation	71
4.3.4 Sufficiency of the Partition Set	73
4.4 Data Set Approximation	77
4.5 Region Identification Phase	78
4.6 Region Refinement Phase	82
4.7 Partition Simplification Algorithm	87
4.8 A Decision Rule Set Example - The Iris Data Set	90
4.9 Adjunct Design Issues	91
4.9.1 Resolving Rule Conflicts	91
4.9.2 Classifying Orphan Data	93
4.10 Toward a Concurrent Version of GRaCCE	94
4.11 Summary	98
V. Properties of GRaCCE	100
5.1 Theoretical Foundations	100
5.1.1 The Bayes Theorem and Related Concepts	100
5.1.2 The Bayes Error	102
5.1.3 Properties of the kNN Algorithm	105
5.1.4 Relevant Definitions	106
5.2 GRaCCE-Related Theorems and Discussion	107

	Page
5.2.1 Theorem Development	108
5.2.2 Discussion	110
5.3 Search Space Considerations	112
5.4 Fitness Landscape Considerations	114
5.5 Time Complexity Analysis	116
5.5.1 Data-Related Assumptions	116
5.5.2 Derivation of Sequential Time Complexity . .	121
5.5.3 Comparison to Decision Tree Algorithms . . .	126
5.5.4 Derivation of Concurrent Growth Rate	127
5.6 Summary	128
VI. Experiments with GRaCCE	129
6.1 Objectives	129
6.2 Selection of Data Sets	129
6.3 Data Preparation	131
6.4 Reporting of Results	133
6.5 Inter-Algorithm Comparison	134
6.5.1 Test Methodology	134
6.5.2 GRaCCE Specific Considerations	135
6.5.3 Choices for System Comparison	136
6.5.4 Algorithm Parameters	136
6.5.5 Classification Accuracy	139
6.5.6 Generalization	139
6.5.7 Decision Rule Set Complexity	142
6.5.8 Viewing the “Big Picture”	147
6.6 Intra-Algorithm Comparison	147
6.6.1 Effect of Classifying Orphan Data	147
6.6.2 Effect of Partition Simplification	156

	Page
6.6.3 Robustness	156
6.7 Assessing cGRaCCE	161
6.7.1 Effect of Concurrency on Run-Time Execution Performance	166
6.7.2 Scalability	167
6.8 Summary	179
VII. Conclusions and Recommendations	182
7.1 Research Summary	182
7.2 Test Summary and Conclusions	184
7.3 Contributions	185
7.3.1 A method for generating a sufficient set of par- titions for decision rule construction.	186
7.3.2 The development of a GA-based “0/1” approach for supervised clustering of data.	187
7.3.3 A method for approximating the training set to accelerate rule induction.	187
7.3.4 The development of a concurrent architecture for GRaCCE.	188
7.4 Recommendations for Future Research	188
7.4.1 Translate GRaCCE from <i>MatLab</i> to C^{++} . . .	188
7.4.2 Develop a Fully Parallel Version of GRaCCE .	188
7.4.3 Adapt for Very Large Databases	189
7.4.4 Optimize GRaCCE for Discrete Data Sets . .	189
7.4.5 Application of Genetic Programming to GRaCCE	190
7.5 The Last Word	190
Appendix A. Review of Genetic Search Methods	191
A.1 What is a Genetic Algorithm?	191

	Page
A.2 Algorithm	192
A.3 Theory	194
A.3.1 Schema	194
A.3.2 Fundamental Theorem	195
A.4 Problem Encoding	196
A.5 Reproduction Strategies	198
A.5.1 Selection	199
A.5.2 Recombination	200
A.5.3 Mutation	202
A.5.4 Evaluation	202
A.5.5 Reconstitution	203
A.6 What is Genetic Programming?	204
Appendix B. GRaCCE User's Guide	207
B.1 Running GRaCCE	207
B.2 Preprocessing Operations	208
B.2.1 Loading the Data File	208
B.2.2 Feature Selection	209
B.2.3 Winnowing	209
B.3 Decision Rule Induction	209
B.4 Helpful Hints	210
B.5 Leaving GRaCCE	212
Appendix C. Sample GRaCCE Report	219
Appendix D. Summary of Mathematical and Algorithmic Notation .	231
Appendix E. Description of cGRaCCE Hardware Configuration . . .	235
Bibliography	237

	Page
Vita	252

List of Figures

Figure		Page
1.	Overview of the data mining process.	3
2.	Data mining techniques can help commanders assess information produced by C ³ I systems.	5
3.	Decision Tree for the Iris Data Set.	12
4.	Generic Decision Tree Induction Algorithm.	13
5.	Univariate Decision trees can yield rule sets that are more complex than necessary.	16
6.	Procedure used by CART for finding splits at decision tree nodes.	18
7.	CART generated splits (solid lines) minimize impurity in a way that is not necessarily optimal (denoted by dotted lines). . . .	20
8.	Pruning can reduce the predicted number of errors (PNE). . .	22
9.	Piecewise Linear Classifier algorithms search for hyper-planes that cut the maximum number of Tomek links separating two classes.	27
10.	No Tomek links are generated in the middle region. This is because each opposing class is not close enough to the other across this divide.	28
11.	Block diagram representation of a GA-based Classifier for Rule Induction.	31
12.	Bayesian Network for Charniak's "Family Out" problem. . . .	39
13.	A simple, fully connected Multi-layer Perceptron (3 layer, 4 input, 2 output).	42
14.	Comparative Feature Saliency.	46
15.	Karhunen-Loève Transform - Original two dimensional feature set is projected onto eigenvector u_1 , which has the greatest eigenvalue.	49

Figure		Page
16.	The EDBFM is the covariance matrix of unit vectors normal to the class decision boundaries.	50
17.	Discrete mode categories created without respect to class. . .	53
18.	Discrete mode categories created with knowledge of class labels.	54
19.	XOR Data Set.	55
20.	Over-fit class boundaries (left) tend to be more complex than those that aren't (right).	55
21.	Normalization can sometimes impair separability.	60
22.	Data Set SYN01 before (above) and after (below) the Winnowing operation.	68
23.	Partitions are generated to separate boundary point pairs. . .	69
24.	Global Partition Generation for Data Set SYN03.	71
25.	The interleaving of class modes in SYN04 renders global partitions useless.	72
26.	Generation process for boundary point pairs.	73
27.	Set of Generated Partitions for SYN01.	74
28.	Set of Generated Partitions for SYN04.	74
29.	Iterative Partition Generation Algorithm.	75
30.	Initial (above) and Final (below) Partition Sets for Data Set SYN02.	76
31.	GA Chromosome Structure.	80
32.	Region Identification Phase Algorithm.	83
33.	Greedy Search Algorithm for the Initial CH Region.	84
34.	SYN01 Partitions selected for Class 1.	84
35.	SYN01 Partitions selected for Class 2.	85
36.	SYN01 Partitions selected for Class 3.	85
37.	Partitions selected for SYN04.	86
38.	Partition Simplification Algorithm.	89

Figure		Page
39.	This plot illustrates how the derived partitions are used to separate the classes for the Iris data set.	92
40.	Relaxing the purity parameter (δ_{min}) makes it possible for CH regions of different class to overlap.	93
41.	Orphan data can be classified by assigning it to the class of the closest CH region in terms of Mahalanobis distance.	94
42.	Flowchart for cGRaCCE Tasks.	97
43.	Parallelization Scheme for Extending cGRaCCE.	98
44.	Relationship of PDFs to Decision Boundaries.	102
45.	Relationship of A Posteriori Probabilities to Decision Boundaries.	103
46.	Piecewise Sequence of Linear Partitions.	107
47.	Combination of Local Partitions Approximate the Bayes Decision Surface.	110
48.	Results of Greedy, Adaptive Walk for the Wine Data Set. . .	115
49.	Two Class Unimodal - Gaussian Distribution.	118
50.	Two Class Unimodal - Uniform Distribution.	118
51.	Two Class Multimodal - Symmetric.	118
52.	Five Class Multimodal - Interleaved.	119
53.	Two Class Multimodal - Checkerboard.	119
54.	Data set configuration for worst case complexity.	121
55.	Simplified Pseudo-code for the Region Identification Phase. .	122
56.	Simplified Pseudocode for the Objective Function.	122
57.	Baseline Comparison - Classification Accuracy for Full Feature Set (Rules Only).	140
58.	Baseline Comparison - Classification Accuracy for Reduced Feature Set (Rules Only).	141
59.	Baseline Comparison - Generalization Performance on Original Feature Set.	143

Figure		Page
60.	Baseline Comparison - Generalization Performance on Reduced Feature Set.	144
61.	Baseline Comparison - Average Rule Set Size for the Original Feature Set.	148
62.	Baseline Comparison - Average Rule Set Size for the Reduced Feature Set.	149
63.	Baseline Comparison - Average Conditions per Rule for the Original Feature Set.	150
64.	Baseline Comparison - Average Conditions per Rule for the Reduced Feature Set.	151
65.	Baseline Comparison - Average Terms per Rule Set (Original Feature Set).	152
66.	Baseline Comparison - Average Terms per Rule Set (Reduced Feature Set).	152
67.	Baseline Comparison - Average Rule Set Compactness (Original Feature Set).	153
68.	Baseline Comparison - Average Rule Set Compactness (Reduced Feature Set).	154
69.	Normalized Comparison - Original Feature Set.	155
70.	Normalized Comparison - Reduced Feature Set.	155
71.	This graph shows the mean error rate reduction (relative to the rules-only results) when orphan data is classified using the Mahalanobis distance metric.	157
72.	Error Rate Changes Resulting from Partition Simplification (Original Feature Set).	158
73.	Reduction in Rule Set Complexity resulting from Partition Simplification (Original Feature Set).	158
74.	Error Rate Changes Resulting from Partition Simplification (Reduced Feature Set).	159
75.	Reduction in Rule Set Complexity resulting from Partition Simplification (Reduced Feature Set).	159

Figure		Page
76.	Effect of γ_{min} on Cancer Data.	162
77.	Effect of γ_{min} on Glass Data.	162
78.	Effect of γ_{min} on Iris Data.	163
79.	Effect of γ_{min} on Syn04 Data.	163
80.	Effect of γ_{min} on Wine Data.	164
81.	Effect of Concurrency - Cancer Data.	168
82.	Effect of Concurrency - Glass Data.	169
83.	Effect of Concurrency - Mushroom Data.	170
84.	Effect of Concurrency - Soybean Data.	171
85.	Effect of Concurrency - SYN03 Data.	172
86.	Effect of Concurrency - SYN04 Data.	173
87.	Effect of Concurrency - Wine Data.	174
88.	Scalability of Region Identification Phase for $q = 10$ as data set complexity increases.	176
89.	Scalability of Region Identification Phase for $q = 100$ as data set complexity increases.	176
90.	Scalability of Region Identification Phase for $q = 1000$ as data set complexity increases.	177
91.	Scalability of cGRaCCE as the size of the Mushroom data set increases ($q = 1000$).	178
92.	Scalability of cGRaCCE as the size of the SYN04 data set increases ($q = 100$).	178
93.	The GA is used to search a given landscape.	192
94.	GA Algorithm Outline.	193
95.	The objective function translates an encoded genotype to its fitness as a solution in the original domain search space. . . .	197
96.	Binary GA Chromosome Format (4 Variable).	198
97.	Roulette Wheel Selection.	199
98.	Stochastic Uniform Selection.	200

Figure		Page
99.	Example of single point crossover.	201
100.	Example of double point crossover.	202
101.	Example of Shuffle Crossover.	203
102.	Example Genetic Program Structure.	205
103.	GRaCCE - Main Menu.	212
104.	GRaCCE - Preprocessing Menu.	214
105.	GRaCCE - Genetic Algorithm Menu.	217
106.	GRaCCE - Region Identification Phase Menu.	218
107.	Architecture of the AFIT Beowulf PC Cluster.	236

List of Tables

Table		Page
1.	C4.5 Summary.	17
2.	CART Summary.	20
3.	OC1 Summary.	25
4.	Piece-wise Linear Classifier Summary.	29
5.	GA-Based Classifier System Summary.	33
6.	Genetic Program based Classifiers Summary.	35
7.	Bayesian Network Summary.	41
8.	Synthetic Data Set Descriptions (2 Feature).	65
9.	Partition Specification for Iris Data.	91
10.	CH Region to Partition Mapping for Iris Data.	91
11.	Worst Case Time Complexity of GRaCCE Phases.	96
12.	Scheme for Parallelizing Candidate GRaCCE Tasks.	98
13.	Complexity Metrics for Example Data Sets.	117
14.	Mapping of Design Goal to Test Objectives.	130
15.	Real Wold Data Set Descriptions.	131
16.	Reduced Data Set Descriptions.	132
17.	Classification Error Rate Descriptions.	134
18.	Test Method Descriptions.	134
19.	GA Parameter Settings.	137
20.	GRaCEE - Region Identification Phase Parameter Settings. .	138
21.	CART Parameter Settings.	138
22.	OC1 Parameter Settings.	139
23.	Description of Rule Set Metrics.	145
24.	Data Set Descriptions.	165
25.	Main Menu - Data Set Informational Fields.	213

Table		Page
26.	Main Menu - Selectable Items.	213
27.	Preprocessing Menu - Selectable Items.	214
28.	Genetic Algorithm Menu - Selectable Items.	215
29.	Region Identification Phase Menu - Selectable Items.	216
30.	Explanations of Fields within the GRaCCE Execution Report.	220
31.	General Notation Summary.	232
32.	Notation Summary - Boundary Points.	232
33.	Notation Summary - Class Partition.	232
34.	Notation Summary - Genetic Algorithm (GA).	233
35.	Notation Summary - Region Identification (RI) Phase.	233
36.	Notation Summary - Region Refinement (RR) Phase.	233
37.	Notation Summary - Partition Simplification Algorithm.	234

Abstract

With the increased proliferation of computing equipment, there has been a corresponding explosion in the number and size of databases. Although a great deal of time and effort is spent building and maintaining these databases, it is nonetheless rare that this valuable resource is exploited to its fullest. The principle reason for this paradox is that many organizations lack the insight and/or expertise to effectively translate this information into usable knowledge. While data mining technology holds the promise of automatically extracting useful patterns (such as decision rules) from data, this potential has yet to be realized. One of the major technical impediments is that the current generation of data mining tools produce decision rule sets that are very accurate, but extremely complex and difficult to interpret. As a result, there is a clear need for methods that yield decision rule sets that are both accurate and compact.

The development of the Genetic Rule and Classifier Construction Environment (GRaCCE) is proposed as an alternative to existing decision rule induction (DRI) algorithms. GRaCCE is a multi-phase algorithm which harnesses the power of evolutionary search to mine classification rules from data. These rules are based on piece-wise linear estimates of the Bayes decision boundary within a winnowed subset of the data. Once a sufficient set of these hyper-planes are generated, a genetic algorithm (GA) based "0/1" search is performed to locate combinations of them that enclose class homogeneous regions of the data. It is shown that this approach enables GRaCCE to produce rule sets significantly more compact than those of other DRI methods while achieving a comparable level of accuracy. Since the principle of *Oc-cam's razor* tells us to always prefer the simplest model that fits the data, the rules found by GRaCCE are of greater utility than those identified by existing methods.

EVOLVING COMPACT DECISION RULE SETS

I. Introduction

Information is simultaneously the most useful and cumbersome byproduct of the computer revolution. With the falling cost and increased proliferation of computing and storage equipment, there has been a corresponding explosion in the number and size of databases. Although a great deal of time and effort is spent building and maintaining these databases, it is nonetheless rare that the full potential of this valuable resource is realized. The principle reason for this paradox is that the vast majority of organizations lack the insight and/or expertise to effectively translate this information into usable knowledge (55).

1.1 What is Data Mining?

In light of these conditions, there exists a clear need for automated methods and tools to assist in exploiting the vast amount of available data. This requirement has led to the development of data mining technology. *Data mining is an umbrella term which describes the process of uncovering patterns, associations, changes, anomalies and statistically significant structures and events in data.* Traditional data analysis is assumption driven in the sense that a hypothesis is manually formed and validated (by statistical means) against the data. In contrast, data mining is discovery driven in that useful patterns are automatically extracted from the data (47). In order to accomplish this task, data mining systems frequently utilize methods from disciplines such as artificial intelligence, machine learning and pattern recognition (149).

The data mining algorithms discussed in this document operate on data sets composed of vectors (instances) of independent variables (or features). For example, a database may describe a group of people in terms of their age, sex, income and

occupation. In this case, age is an example of a feature and each instance corresponds to a distinct individual. The general data mining process is described in Figure 1; the subprocesses depicted in this flowchart include:

- Data Selection/Sampling - The sheer size of some databases make it impractical to process them in their entirety. As a result, it is often necessary to winnow the data in some manner or randomly select a subset of instances for processing.
- Cleaning/Preprocessing - During this phase, the selected data is prepared for processing by the data mining algorithm. This can involve translating the data into an acceptable format or replacing missing or illegitimate entries.
- Transformation/Reduction - The purpose of this phase is to revise and/or redefine the feature set. In many cases all the features included in a given data set are not required for prediction. In other instances, it may be desirable to create new features to facilitate the mining process.
- Data Mining - This refers to the application of the selected data mining method to the data.
- Evaluation Criteria - In this phase the output of the mining algorithm is evaluated against a goodness criteria. This is typically done to reduce the volume of information produced to that which is most useful or relevant.
- Visualization - This is the task of massaging the output to facilitate manual analysis. Information that can be easily understood has the best chance of becoming usable knowledge.

While the flowchart portrays a single thread, in reality the data mining process is best characterized as iterative and repetitive. This is because locating information of interest often requires that data be analyzed using several different methods over a series of trials (as one anonymous researcher puts it, “you torture the data till they confess”).

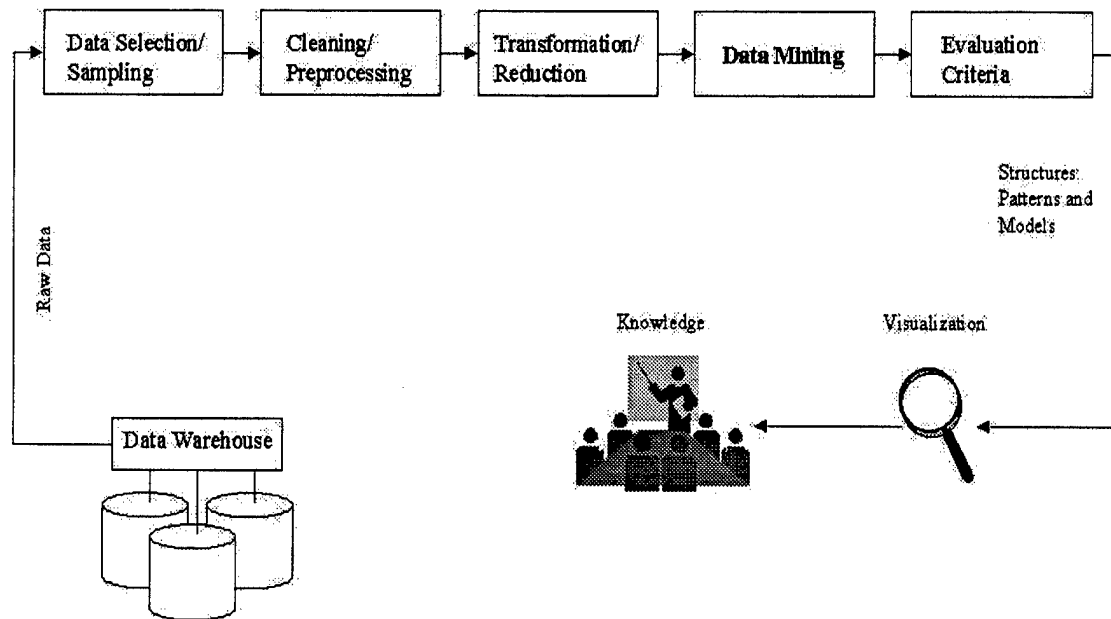


Figure 1 Overview of the data mining process.

To discover the hidden patterns in data, it is essential to build a model consisting of independent variables that can be used to determine a dependent variable (also known as class). Building such a model therefore consists of identifying the relevant independent variables and minimizing the predictive error (125). It is also highly desirable to find the simplest possible model that fits the data, since these are typically the most meaningful and easiest to interpret. This last requirement reflects the principle of *Occam's Razor* which tells us to prefer the simplest model that fits the data (10).

Before we proceed further, it is important to distinguish data mining from pattern recognition as these terms are sometimes confused with each other. Pattern recognition is primarily concerned with the construction of accurate classifiers. A classifier is fundamentally a mapping between a set of input variables x_1, \dots, x_d to an output variable y whose value represents the class label $\omega_1, \dots, \omega_m$ (10). In general, representing the knowledge embodied within the classifier structure is not a priority. Consequently, while there is no shortage of extremely accurate classifiers, some of the best are akin to a *black box*; that is, they give little or no insight into *why* they make decisions. Multi-layer Perceptrons (MLPs) (10; 86) exemplify these types of systems because its classification rules are embedded in its structure. Since MLP components (node activation functions, connection weights, etc.) encode complex mathematical functions, articulating the rules they represent is a difficult problem (84).

In contrast, the primary purpose of data mining is not simply classification, but to provide meaningful knowledge to the user regarding the classification process. Thus, the models produced by data mining algorithms should be in a form that lends itself to analysis by the user. Decision rule sets which linearly partition the data space into class homogeneous regions meet this requirement. Examples of techniques that accomplish decision rule induction (DRI) from data include decision trees (11; 94; 105) and genetic algorithm (GA) based Classifier Systems (41; 60; 127).

1.2 *Applicability of Data Mining to the USAF Mission*

While the vast majority of data mining “success stories” have a distinctly commercial flavor (7), the general purpose nature of this technology makes it equally applicable to military organizations. Accordingly, we discuss some potential ways that predictive data mining methods can aid the USAF mission.

1.2.1 *Augmenting the OODA Loop.* The Observation, Orientation, Decision, Action (OODA) loop is perhaps the most widely accepted model of the bat-

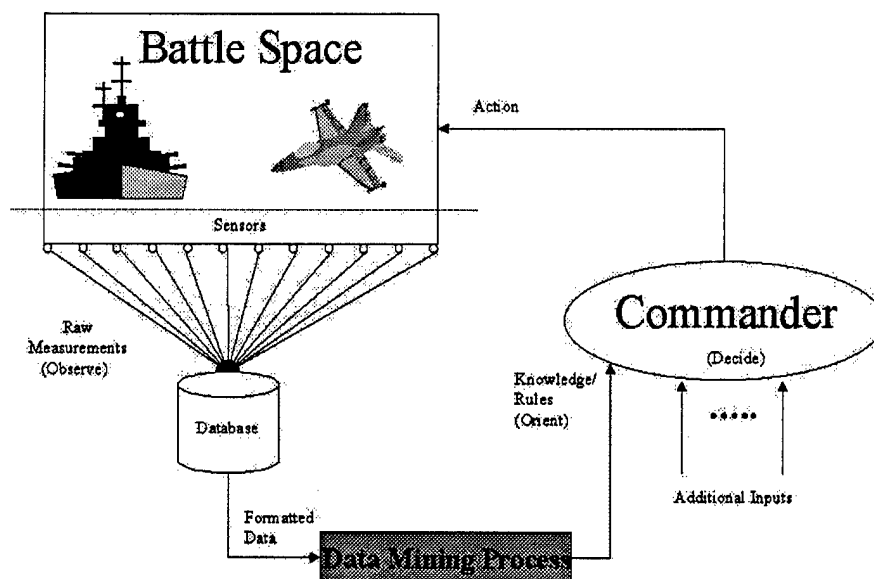


Figure 2 Data mining techniques can help commanders assess information produced by C³I systems.

tlefield decision process (133). While command, control, communication, computer and intelligence (C⁴I) systems are essential to victory on the modern battlefield, they also routinely flood commanders with data during an engagement. As a result, tools which help commanders make informed decisions in a data intensive environment are at a premium. Ideally, these tools should be able to synthesize a diverse range of information into knowledge or rules which can be easily understood by a commander (or his staff). For example, data mining software can (over time) generate rules that distinguish a full-scale attack from a spoiling attack based on disparate sensor measurements. Figure 2 shows how the OODA loop can be modified to incorporate data mining capabilities.

1.2.2 Defense Against Cyber-Attack. The specter of Information Warfare (IW) looms large as the 21st century approaches. The USAF (143) defines IW as

“Any action to deny, exploit, corrupt or destroy the enemy’s information and its functions while protecting Air Force assets against those actions and exploiting its own military information operations.” According to this definition, the defense of USAF resources against hacker intrusion attempts is a key IW activity. One way to do this in an automated fashion is to record metrics that monitor the activities of users and store them in a database. Given a set of such data containing examples of illegal and authorized users, data mining techniques can be utilized to discover rules describing abnormal behavior. These rules can alert system administrators to the presence of potentially hostile users. A similar approach could also be used to automatically generate decision rules to detect the presence of computer viruses.

1.2.3 Weather Prediction. Weather prediction is a critical Air Force support function. Because of this, volumes of data exist which document meteorological conditions at Air Force bases worldwide. Data mining techniques could be applied here in order to generate a more reliable set of decision rules for predicting weather phenomena. For instance, decision rules can be generated to predict go/no-go conditions at a given launch site, hours ahead of a scheduled mission based on data from surrounding weather stations and sensors. Such an approach has the potential to improve safety, preserving both lives and USAF resources.

1.2.4 Additional Applications. Because the USAF mission is so broad, there are an almost unlimited variety of other applications of data mining, including identification of recruitment patterns, screening of intelligence intercepts for relevance, and automatic target recognition.

1.3 Research Problem

Ideally, the knowledge discovered by predictive data mining algorithms should provide insight into the problem domain (in addition to accurately classifying the data). This requirement is especially valid for military or intelligence problems,

where such insight can have life or death consequences. Given these stakes, a fundamental question to ask is: is it realistic to expect automated systems (by themselves) to generate knowledge of such importance? Judging from the level of sophistication found in the current generation of data mining tools, the answer to this question is clearly no (47). As a result, people are still required to perform the tedious task of analyzing the output of these tools to identify knowledge which is interesting or useful.

A common sense approach to streamlining this process is to develop tools that mine decision rule sets which are meaningful and easy to interpret. As previously discussed, predictive models with the following characteristics tend to fall into this category:

- Accurate - Classify data with a low error.
- Compact - Use a minimum number of rules.
- Simple - Individual rules should have a low level of complexity.

While a wide variety of DRI techniques exist, these often fail to yield rule sets that achieve these desired attributes. When these methods are surveyed in Chapter II, it will be shown that a priori assumptions regarding the data and rule structure made by these algorithms is a major contributor to this problem. Although these assumptions help to make the algorithms more efficient, they can also cause the quality of the decision rules to suffer. Thus, designing a DRI method which avoids these pitfalls is a valid research problem.

1.4 Approach and Design Goals

We propose the development of the Genetic Rule and Classifier Construction Environment (GRaCCE) as an alternative to existing DRI algorithms. GRaCCE is a multi-phase algorithm which harnesses the power of evolutionary search to mine classification rules from data. These rules are based on piecewise linear estimates

of the Bayes decision boundary (81) within a winnowed subset of the data. Once a sufficient set of these hyper-planes are generated, an evolutionary search is performed to locate combinations of them which enclose class homogeneous (CH) regions of the data. In particular, the objective function for this search seeks CH regions that meet a threshold level of purity (classification accuracy with respect to the majority class) while maximizing coverage of the target class and minimizing region complexity (in terms of the number of defining partitions). These regions are refined further and used to generate a final rule set for classifying the data. In order to successfully solve the problem outlined above, the algorithm must satisfy the following design goals:

- Function as a general purpose DRI algorithm, capable of processing a variety of different types of data sets without any prior knowledge of the problem. In particular, the system must be able to successfully process data sets that vary with respect to number of classes, features, modes and instances.
- Generate decision rule sets that are compact and simple with respect to those produced by other DRI methods.
- The algorithm must achieve a level of accuracy on a par with other DRI techniques when given the same data.
- The system must be robust with respect to user selectable parameters.
- The algorithm's architecture can be easily parallelized to facilitate scalable run-time execution performance.

1.5 Assumptions

Every data mining method makes some assumptions regarding the data it processes. In the case of GRaCCE, these assumptions include the following:

1. The data is in flat file format, with columns representing independent variables and rows representing separate instances. Data can be extracted from relational databases (RDBs) in this format through the appropriate queries.

2. The data set describes a supervised classification problem, with each instance assigned to a single, labeled class.
3. All instances have the same dimensionality; this requires that missing values have been filled in during preprocessing.
4. The data corresponding to each class resides in one or more clusters (modes).
5. The degree of overlap between any two classes in the data does not make them indistinguishable from each other.
6. The data has been preprocessed to remove temporal or sequential relationships between instances.
7. Variable types are limited to discrete or continuous numeric values. In the case of discrete data types, this involves assigning a distinct natural integer to represent each category. Descriptive text variables (i.e., name) should not be present in the input data.

1.6 *Summary*

This chapter presented the motivation behind and the focus of this dissertation; subsequent chapters detail how the goals introduced here are satisfied. Chapter 2 covers background on existing approaches to the problem of decision rule induction. Chapter 3 summarizes fundamental classification and pattern recognition issues which must be dealt with by any rule induction algorithm. Chapter 4 provides a comprehensive description of how GRaCCE processes a given data set. Chapter 5 describes the theoretical underpinnings of GRaCCE. In addition, a model for the algorithm's average and worst case time complexity is derived. Chapter 6 documents the methodology for testing the system and the corresponding results. This material includes a comparison of the GRaCCE to several decision tree algorithms. Lastly, Chapter 7 provides a summary of the research, its significant contributions, and recommendations for future work.

II. Background

Induction is the process by which general conclusions (or rules) are learned from specific facts. There is a rich and diverse body of work addressing the problem of automating the induction process. Indeed, important advances in this area have been made by researchers in the artificial intelligence, machine learning, pattern recognition and statistical fields (to name a few). For data mining problems, logical induction is performed on particular data sets. In this problem domain, we want to derive decision rules to assign instances from a data set to a category (or class). In this chapter, a cross-section of rule induction methods are examined with an eye toward identifying the characteristics that make each data mining approach unique. This discussion (and the one in the following chapter) furnishes the necessary background and context to evaluate the GRaCCE algorithm and compare it to others.

2.1 Introduction to Decision Rule Induction

At its core, decision rule induction is a *search* for regularities within the data set that can be used to organize it into a finite, predetermined set of classes. Consequently, while one induction method can be very different from another, they can be described using a common template consisting of the following characteristics:

- Search Method(s) - As the name implies, this describes the type of search used by the induction algorithm. Examples of search methods include depth-first, evolutionary, and gradient descent.
- Search Heuristic - The search heuristic is the metric the search method seeks to optimize. For example, in a genetic algorithm, a heuristic is used by the objective function to rate the fitness of a given solution.
- Data Related Assumptions - This category refers to assumptions the induction method makes about the structure of the data. For example, does the technique assume the data is discrete or continuous?

- Rule Related Assumptions - These are assumptions made about the structure of the induced rule set. For example, are the conditions in the rule based on single (univariate) or multiple (multivariate) data attributes?

In subsequent sections, each of these characteristics is highlighted when discussing each rule induction technique. This information is intended to help the reader develop a balanced perspective on how each method is unique.

2.2 Decision Trees

Decision trees are hierarchical, sequential classification structures that recursively partition a set of objects (data). The tree has three basic components: decision nodes, branches and terminal nodes (or leafs). The decision node performs a mathematical or logical test on the data attributes. The purpose of the test is to unambiguously partition (split) the data in some fashion. Once the test is performed, the data is forwarded to the appropriate child node. Each decision node (parent) has two or more child nodes. Branches connect the parent to its child nodes; each branch corresponds to a distinct outcome of the test performed at the parent node. A leaf is a childless node which corresponds to a class label. To be viable, a tree must contain zero or more decision nodes and one or more leaf nodes.

An example of a decision tree (based on the Iris data set (34)) is shown in Figure 3. A feature vector can be classified by a decision tree by starting at the root node and executing the test at each decision node (taking the appropriate branch) until a leaf is encountered. Note that the particular test associated with each branch is described by its arc label. The data sample is then assigned to the class associated with the leaf. The leaf then indicates the class decision for the data sample. Thus a path from the root node to a leaf can be thought of as a decision rule.

2.2.1 Fundamentals of Decision Tree Induction. Decision tree induction (DTI) is the process of building a tree from a given data set. Almost all methods use

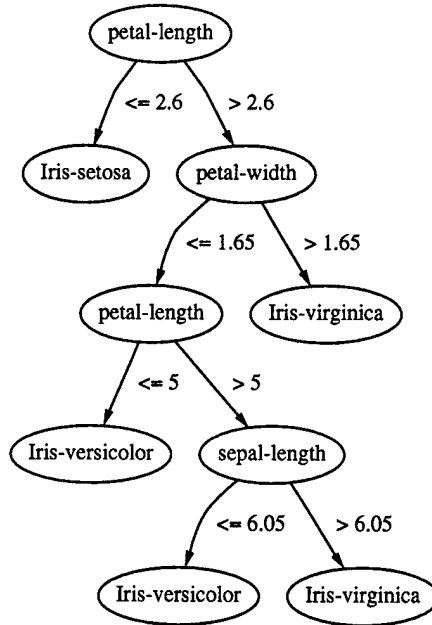


Figure 3 Decision Tree for the Iris Data Set.

a sequential divide and conquer approach: starting with a set of training cases (Ω), perform successive partitioning of cases until all subsets belong to a single class. The primary task at each decision node is determining how to partition the remaining data. For reference purposes, a generic tree induction algorithm is shown in Figure 4.

While a tree that minimizes size and maximizes accuracy can be constructed through the application of exhaustive search or dynamic programming, these approaches usually are computationally infeasible except for trivial data set (111). As a consequence, most DTI algorithms use a search which is greedy in nature in order to improve their time complexity. This means that the splits made at each decision node are based on what appears best for that particular node; in short, partitions are selected based on local, rather than global, optimization criteria. Although the greedy approach can be augmented with look-ahead algorithms, these schemes are considered inferior to the simple greedy heuristic because they have a large computational cost but yield no significant improvement in tree quality (94).

```

procedure PARTITION( $\Omega$ )
  where:  $\Omega$  is a set of class labeled training instances.
  1. If all instances in  $\Omega$  belong to the same class, then return.
  2. Find the most discriminatory, or significant feature.
  3. Partition the entire set,  $\Omega$ , located at the root of the tree, into several
     subsets using the found feature. The number of child nodes originating
     from the root node will be equal to the number of possible values the
     selected feature can take on.
  4. For each child node, call PARTITION( $\Omega_i$ ), where  $\omega_i$  is the subset
     of  $\Omega$  belonging to the  $i$ th child node.

```

Figure 4 Generic Decision Tree Induction Algorithm.

Starting with Hunt's Concept Learning System (62), a great deal of research has been devoted to the problem of growing decision trees. The resulting refinement of decision tree theory has helped to make it one of the most commonly used data mining techniques. The popularity of decision trees is due to three basic properties; these are:

- They produce rules that people can understand. This is primarily because the rules have logical conditions and the data they cover are tightly clustered together.
- Rules can be generated relatively quickly due (in part) to the use of greedy search algorithms.
- The generated rules tend to exhibit good classification accuracy and generalization (due to pruning of the tree).

Perhaps the most important task in building a decision tree is determining how to partition the data at each node. Because of this, DTI algorithms are frequently categorized by the type of hyper-plane used to split the data space. The two most fundamental of these categories, univariate and oblique, are discussed in the sub-

sections that follow. This background is augmented with a discussion of splitting criteria and tree pruning techniques. Lastly, some of the newer approaches to this problem are surveyed.

2.2.2 Univariate Partition Approach. As the name implies, univariate (or axis-parallel) partitions linearly divide the data into classes using a single attribute, x_i . Such partitions take the form: $x_i \leq C$, where C is a constant threshold. Given this model, finding a split entails selecting the single feature that acts as the best discriminator for the data present at each decision node.

Quinlan developed the two best known algorithms in this category: the Interactive Decotimizer 3 (or ID3) (104) and its successor, C4.5 (105). These algorithms are similar in that they follow the procedure outlined in Figure 4. The principal difference between them lies in the criteria utilized for split selection. For ID3, the most discriminatory feature chosen is the one that maximizes *information gain*. This is based on Shannon's definition of information entropy (115) which computes the expected amount of information (in bits) needed for class prediction. Equation 1 calculates the entropy for a set of data S consisting of m classes and d features. Within set S , ω_j represents the subset of data belonging to the j th class. Note that entropy is minimized when S consists of a single class; in contrast, it is maximized when each class is equally represented in the set.

$$\text{Entropy}(S) = - \sum_{j=1}^m \frac{|\omega_j|}{|S|} \log_2 \left(\frac{|\omega_j|}{|S|} \right) \quad (1)$$

The entropy for a particular feature (x_i) in set S is given by

$$\text{Entropy}(S, x_i) = \sum_{k=1}^v \frac{|S_k|}{|S|} \text{Entropy}(S_k) \quad (2)$$

The composite entropy for each feature is summed over the individual entropies for each of the possible *values* that can be taken by a given feature. Thus, computing the entropy requires the range of values for each feature be categorized in some manner. In this context, let v be the number of categories for the i th feature and S_k be the subset of data in S where feature x_i is assigned to the k th category. Note that this approach can be problematic for continuous valued attributes if the discrete values are arbitrarily assigned. In turn, the *information gain* criteria measures the effectiveness of a particular feature in reducing the entropy for the set.

$$\text{Information Gain}(S, x_i) = \text{Entropy}(S) - \text{Entropy}(S, x_i) \quad (3)$$

While effective, ID3 tends to produce relatively large trees. The reason for this is a bias in favor of tests with many outcomes (17). In order to remedy this problem, Quinlan developed C4.5 (summarized in Table 1). The *gain ratio* search criterion used by C4.5 is shown below.

$$\text{Split Information}(x_i) = - \sum_{k=1}^v \frac{|S_k|}{|S|} \log_2 \left(\frac{|S_k|}{|S|} \right) \quad (4)$$

$$\text{Gain Ratio}(x_i) = \frac{\text{Information Gain}(S, x_i)}{\text{Split Information}(x_i)} \quad (5)$$

By maximizing the gain ratio, priority is given to attributes that best distinguish between classes $\omega_1, \dots, \omega_m$ while minimizing the depth of the tree. Because attributes that exhibit these properties are located closer to the root node, this selection process results in smaller decision trees.

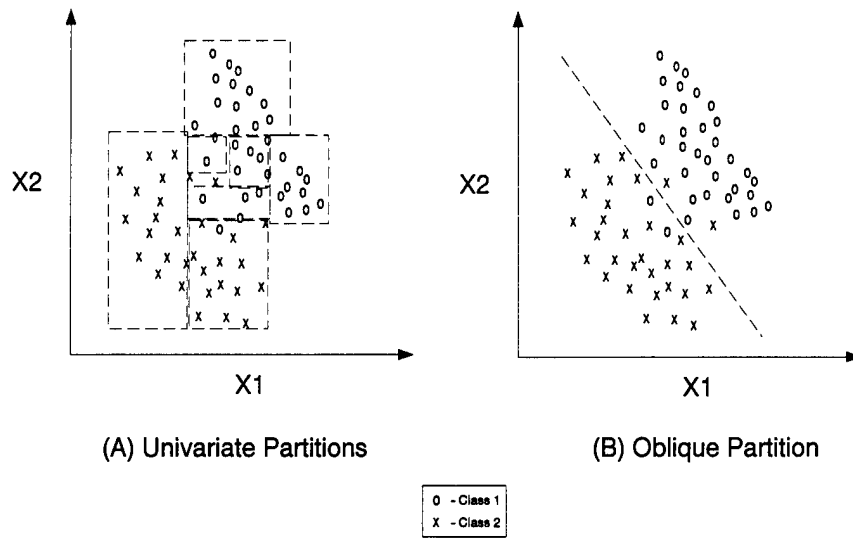


Figure 5 Univariate Decision trees can yield rule sets that are more complex than necessary.

Univariate partitioning methods are attractive because they are straightforward to implement (since only one feature is analyzed at a time) and the derived rule set is relatively easy to understand. The main disadvantage, however, is that the rule set is constrained in how it can represent the actual structure of the data. This situation is illustrated by Figure 5. Note that while the data set can be separated relatively easily, the requirement that the partitions be univariate limits the solutions that can be considered. As a result, axis-parallel splits tend to produce deep trees that yield rule sets which are more complex than necessary (57). Another complicating factor is the mutual exclusive nature of decision tree rules. Although such decision rules unambiguously classify the data, the extra logic required causes the tree to grow much larger as compared to rule sets that tolerate overlap (149).

2.2.3 Oblique Partition Approach. Oblique partitioning provides a viable alternative to univariate methods. Unlike their univariate counterparts, oblique partitions are formed by *combinations* of features. The general form of an oblique partition is given by

Table 1 C4.5 Summary.

Characteristic	Description
Search Method	Deterministic search using a greedy heuristic.
Search Criteria	Gain Ratio
Data Related Assumptions	Heuristic evaluates discrete data. Continuous data is categorized.
Rule Related Assumptions	Each condition corresponds to a univariate, linear partition.

$$\sum_{i=1}^d \beta_i x_i \leq C \quad (6)$$

where β_i represents the coefficient of the i th feature. Because of their multivariate nature, oblique methods offer far more flexibility in partitioning the data space; this flexibility comes at a price, however. Consider that given a data set containing n instances of dimension d , there can be $2 \times \sum_{i=0}^d \binom{n-1}{i}$ oblique splits if $n > d$ (140); each split is a hyper-plane that divides the search space into two non-overlapping halves. For univariate splits, the number of potential partitions is much lower, but still significant, $n \times d$ (94). In short, finding the *right* oblique partition is a difficult task.

Given the size of the search space, choosing the right search method is of critical importance in finding good partitions. Perhaps the most comprehensive reference on this subject is the landmark book by Breiman (et al) on classification and regression trees (CART). In it, Breiman describes his CART algorithm for constructing oblique decision trees. At a macro level, CART uses the same basic algorithm shown in Figure 4; it also uses a generic goodness metric for measuring the split quality. At the decision node level, however, the algorithm (summarized in Figure 6) becomes extremely complex. CART starts out with the best univariate split. It then

```

To induce an oblique split at node  $T$  of the decision tree:
  Normalize values for all  $d$  attributes.
   $L = 0$ ;
  WHILE(TRUE)
     $L = L + 1$ ;
    Let the current split  $s_L$  be  $v \leq c$ , where  $v = \sum_{i=1}^d a_i x_i$ .
    FOR  $i = 1, \dots, d$ 
      FOR  $\gamma = -0.25, 0, 0.25$ 
        Search for the  $\delta$  that maximizes the goodness
        of the split  $v - \delta(a_i + \gamma) \leq c$ .
      Let  $\delta^*, \gamma^*$  be the settings that result in the highest goodness
      for these three searches.
       $a_i = a_i - \delta^*, c = c - \delta^* \gamma^*$ .
    Perturb  $c$  to maximize the goodness of  $s_L$ , keeping  $a_1, \dots, a_d$  constant.
    IF  $|\text{goodness}(s_L) - \text{goodness}_{L-1}| \leq \epsilon$  THEN break;
    Eliminate irrelevant attributes in  $a_1, \dots, a_d$  using backward elimination.
    Convert  $s_L$  to a split on the unnormalized attributes.
    Return the better of  $s_L$  and the best axis-parallel split as the split for  $T$ .

```

Figure 6 Procedure used by CART for findings splits at decision tree nodes.

iteratively searches for perturbations in feature values (one feature at a time) which maximize some goodness metric. At the end of the procedure, the best oblique and axis-parallel splits found are compared and the better of these is selected.

In the above algorithm, the “goodness” criteria is intentionally left undefined. CART was designed to accommodate a number of possible criteria for split evaluation. In general, one wants the leaves of a tree to be *pure* with respect to a given class. This requires the DTI algorithm to ensure each child node has less impurity than its parent. Breiman (11) proposed an impurity function $i(S)$ (known as the Gini Index) which is defined as

$$i(S) = \sum_{i \neq j} p(\omega_i|S)p(\omega_j|S) \quad (7)$$

where $p(\omega_i|S)$ is the probability of a random sample X belonging to class ω_i given the distribution of data in set S . Using this function, the goodness of a split can be defined in terms of the decrease in impurity. Such a metric is shown in Equation 8 where L and R denote the opposite sides of the hyper-plane (h) under consideration. Thus, p_R is the proportion of S that goes to the right child and $i(S_R)$ is the corresponding impurity of that subset. Thus using $\Delta i(h, S)$ as a goodness criteria insures that the partitions resulting in the greatest decrease in impurity are chosen.

$$\Delta i(h, S) = i(S) - i(S_R)p_R - i(S_L)p_L \quad (8)$$

Although CART provides a powerful and efficient solution to a very difficult problem, it is not without its disadvantages. For example, because it is fully deterministic, it has no inherent mechanism for escaping from local optima (i.e., the best solution found). As a result, CART has a tendency to terminate its partition search at a given node too early. Perhaps the most fundamental disadvantage of CART (and of decision trees in general) is that the DTI process can cause the metrics to produce misleading results. In particular, because DTI algorithms choose what is locally optimal for each decision node, they inevitably ignore splits which score poorly alone, but yield better solutions when used in combination. This problem is illustrated by Figure 7. The solid line indicate the splits found by CART; the order of induction is reflected by each split's numeric label. Although each split optimizes the impurity metric, the end product clearly does not reflect the best possible partitions (indicated by the dotted lines). However, when evaluated as individuals, the

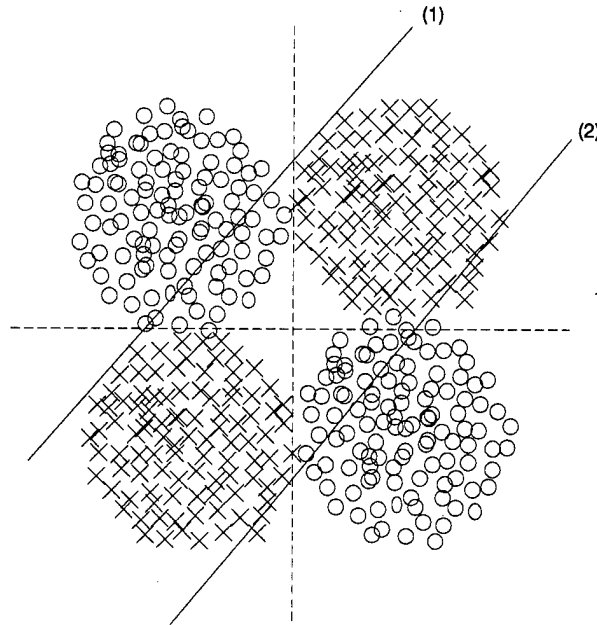


Figure 7 CART generated splits (solid lines) minimize impurity in a way that is not necessarily optimal (denoted by dotted lines).

dotted lines register high impurities and are therefore not chosen. Given this, it is apparent that the sequential nature of decision trees can prevent the induction of rule sets which reflect the natural structure of the data.

2.2.4 Pruning the Tree Structure. Another problem with the induction process is that it is difficult to determine when to stop. This problem manifests itself in the form of decision trees which *over-fit* the data; that is, they infer more structure than is justified by the training cases. This is especially true if the criteria

Table 2 CART Summary.

Characteristic	Description
Search Method	Deterministic search using a greedy heuristic.
Search Criteria	Impurity
Data Related Assumptions	Data can be discrete or continuous.
Rule Related Assumptions	CART uses linear partitions which can be either univariate or oblique.

for halting growth is to have every training instance correctly classified. One solution to the problem of over-fitting is to reduce or limit the size of the decision tree in some manner. In general, techniques which accomplish this are divided into preemptive or pruning categories, depending on where the procedure is applied during the DTI process (104).

Preemptive methods use some predetermined criteria to halt tree growth. These criteria are typically tested on a per-node basis; as a result, growth may halt along one branch of the tree, but continue along others. For example, a node may stop generating children when its assigned data meets a given purity threshold or falls below a fixed size threshold. The problem with this approach is that choosing the stopping criteria arbitrarily may cause the tree to over-fit or under-fit (due to premature halting of the DTI algorithm) the data.

In contrast, the more commonly used approach is to first generate a tree that over-fits the data and then apply a pruning technique to reduce the size of the tree (111). Pruning involves the removal of subtrees that do not contribute significantly to classification accuracy due to poor generalization. Quinlan's *pessimistic* pruning technique (105) provides a good example of the utility of these methods. Conceptually, Quinlan's approach is quite simple: subtrees are pruned whenever doing so causes a reduction in the predicted number of errors (PNE). While the number of classification errors cannot be computed exactly, Quinlan estimates them using the binomial distribution (59) for a given confidence level. PNE is computed for a given subtree using Equation 9 where:

- N_i is the number of training cases in the i th branch of the subtree,
- E_i is the number of training classes incorrectly classified in that branch,
- U_{CF} is the binomial distribution with parameters E, N and confidence level CF .

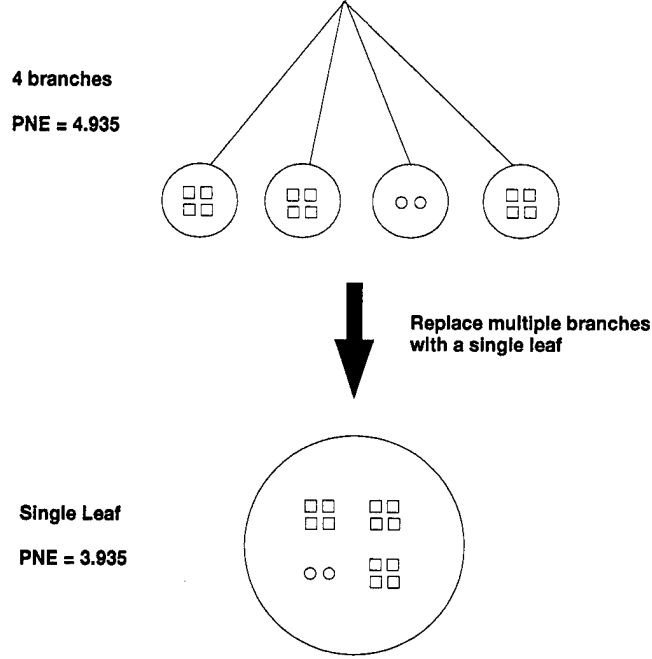


Figure 8 Pruning can reduce the predicted number of errors (PNE).

$$\text{PNE} = \sum_{\text{Branches}} N_i \times U_{CF}(E_i, N_i) \quad (9)$$

Because it is summed over all branches of the subtree, it is possible to reduce the PNE by replacing a complex error free subtree with a simpler structure (such as a leaf) that contains some error. For example, consider the subtree shown in Figure 8 which has four branches (containing 14 training instances) and no errors. The PNE for this branch is

$$\begin{aligned} \text{PNE}_{\text{Before}} &= 3 \times (4 \times U_{25}(0, 4)) + 2 \times U_{25}(0, 2), \\ &= 3 \times (4 \times 0.63) + 2 \times 0.5625 = 4.935. \end{aligned}$$

If this subtree were to be replaced by a single leaf with two errors (since the items in one branch are now misclassified), the resulting PNE is

$$\begin{aligned}\text{PNE}_{\text{After}} &= 14 \times U_{25}(2, 14), \\ &= 14 \times 0.2811 = 3.935.\end{aligned}$$

As the above example illustrates, simplifying the tree structure lowers the PNE even though the change results in more training errors. While this may seem like a paradox, it is indicative of how over-fitting can degrade a decision tree's ability to generalize. When the structure is simplified, the expected generalization (and accuracy) improve. While there are many other pruning techniques in this category, studies have shown that no one method is generally superior to the others (93).

2.2.5 Variant Methods. Although algorithms such as ID3, C4.5 and CART make up the foundation of DTI practice, there is always room for improvement in terms of the accuracy, size, and generalization ability of the generated trees. As would be expected, many researchers have tried to build on the success of these techniques by developing better variations of them. In this section, several of these variant algorithms are surveyed.

2.2.5.1 Split Selection using Random Search. Since random search techniques have proven extremely useful in finding solutions to non deterministic polynomial complete (NP-complete) problems (95), it is natural they be applied to DTI. Heath (54) developed a DTI algorithm called SADT ¹ which uses a Simulated Annealing (SA) process to find oblique splits at each decision node. SA is a variation of hill climbing which, at the beginning of the process, allows some random downhill moves to be made (109). As a computational process, SA is patterned after the physical process of annealing (71), in which metals are melted (at high temperatures) and then gradually cool until some solid state is reached.

Starting with an initial hyper-plane, SADT randomly perturbs the current solution and determines the goodness of the split by measuring the change in impurity

¹Simulated Annealing of Decision Trees

(ΔE). If ΔE is negative (i.e., impurity decreases), the new hyper-plane becomes the current solution; otherwise, the new hyper-plane becomes the current split with probability $e^{-(\Delta E/T)}$ where T is the temperature of the system. Because SA mimics the cooling of metal, its initially high temperature falls with each perturbation. At the start of the process, the probability of replacing the current hyper-plane is nearly 1. As the temperature cools, it becomes increasingly unlikely that worse solutions are accepted. When processing a given data set, Heath typically grows hundreds of trees, performing between 3000 and 30000 perturbations per decision node. Thus, while SADT has been shown to find smaller trees than CART, it is very expensive from a computational standpoint (54).

A more elegant variation on Heath's approach is the OC1 system (described in Table 3) developed by Murthy (94). Like SADT, OC1 uses random search to find the best split at each decision node. The key difference is that OC1 rejects the brute force approach of SADT, using random search only to improve on an existing solution. In particular, it first finds a good split using a CART-like deterministic search routine (see Figure 6). OC1 then randomly perturbs this hyper-plane in order to decrease its impurity. This step is a way of escaping the local optima in which deterministic search techniques can be trapped. If the perturbation results in a better split, OC1 resumes the deterministic search on the new hyper-plane; if not, it re-perturbs the partition a user-selectable number of times. When the current solution can be improved no further, it is stored for later reference. This procedure is repeated a fixed number of times (using a different initial hyper-plane in each trial). When all trials have been completed, the best split found is incorporated into the decision node. This combination of deterministic and limited random search has proven extremely effective in inducing trees that are relatively small and accurate (as compared to methods such as CART). In addition, because the random search is applied in a more focused manner, OC1 is more computationally efficient than SADT.

Table 3 OC1 Summary.

Characteristic	Description
Search Method	Uses a combination of deterministic and random search.
Search Criteria	Impurity
Data Related Assump- tions	Data can be discrete or continuous.
Rule Related Assump- tions	Primarily utilizes oblique (multivariate), linear parti- tions.

2.2.5.2 Incremental Decision Tree Induction. Up to this point, the DTI algorithms discussed grow trees from a complete training set. For serial learning tasks, however, training instances may arrive in a stream over a given time period. In these situations, it may be necessary to continually update the tree in response to the newly acquired data. Rather than building a new decision tree from scratch, the incremental DTI approach revises the existing tree to be consistent with each new training instance. Utgoff (144) implemented an incremental version of ID3 (called ID5R). ID5R uses an *E-Score* criteria to estimate the amount of ambiguity in classifying instances that would result from placing a given attribute as a test in a decision node. Whenever the addition of new training instances does not fit the existing tree, the tree is recursively restructured such that attributes with the lowest E-Scores are moved higher in the tree hierarchy. In general, Utgoff's algorithm yields smaller trees compared to methods like ID3, which batch process all training data. Techniques similar to ID5R include an incremental version of CART, developed by Crawford (21). Incremental DTI techniques result in frequent tree restructuring when the amount of training data is small, with the tree structure maturing as the data pool becomes larger.

2.2.5.3 Decision Forests. Regardless of the DTI method utilized, subtle differences in the composition of the training set can produce significant variances in classification accuracy. This problem is especially acute when cross-validating small data sets with high dimensionality (28) (also refer to Section 3.1).

Researchers have reduced these high levels of variance by using *decision forests*, composed of multiple trees (rather than just one). Each forest is unique because it is grown from a different subset of the same data set. For example, Quinlan's windowing technique (105) induces multiple trees, each from a randomly selected subset of the training data (i.e., a window). Another approach was devised by Ho (57), who based each tree on a unique feature subset. Once a forest exists, the results from each tree must be combined to classify a given data instance. Such committee-type schemes for accomplishing this range from using majority rules voting (53) to statistical methods for combining evidence (119). While these methods can improve overall classification accuracy, it does not yield a single, coherent set of rules. This characteristic makes committee methods undesirable from a data mining perspective.

2.3 Piecewise Linear Classifiers

One of the principal disadvantages of DTI algorithms is that the partitioning metrics make inherent assumptions about the structure of the data set. If these assumptions turn out to be false, the metric may not yield the intended result (recall the example of Figure 7). Piecewise linear classifiers (PLCs) are an alternative that offer a much more flexible approach. These methods locate linear partitions within the data that approximate the *natural* class boundaries within the data. Once found, such hyper-planes serve as building blocks upon which to base classification rules. The fact that class boundaries in "real-world" data sets often overlap make this very difficult, however.

A novel approach to this problem was developed by Park and Sklansky (99; 100), based on earlier research by Sklansky and Michelotti (123). Their method involves the cutting of straight line segments, called Tomek links (139), that join opposing points of different classes in d -dimensional space. These links are generated using a variation on the Condensed Nearest Neighbor (CNN) algorithm (52)

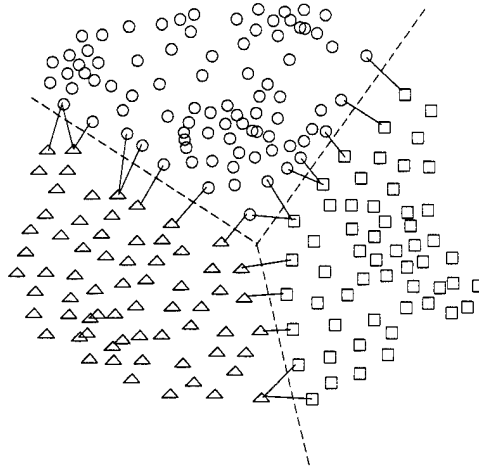


Figure 9 Piecewise Linear Classifier algorithms search for hyper-planes that cut the maximum number of Tomek links separating two classes.

developed by Tomek (139). The basic idea is to link adjacent pairs of points (from different classes) that lie along their common decision boundary.

Each hyper-plane that cuts a Tomek link is akin to a class partition. The quality of the partition increases with the number of links it cuts (assuming all links separate the same two classes). Minimizing the number of hyper-planes required to cut all of the links results in a set of partitions that approximate the Bayes optimal decision surface². Park and Sklansky outline a two phase, iterative training procedure for individual hyper-planes. In the first phase, the hyper-plane is trained such that only those Tomek links connecting two pre-designated classes are cut. During the second phase, the partition orientation is adjusted as to cut Tomek links of other classes (without undoing the cuts made in the first phase). Figure 9 depicts the links and associated partitions generated by the procedure for a synthetic data set.

Piecewise Linear Classifiers are significant in the context of this dissertation because they form decision rules from combinations of hyper-planes which approximate the natural class boundaries. As we show in Chapter IV, aspects of this

²Refer to Chapter V for a detailed discussion of what constitutes a Bayes optimal decision surface.

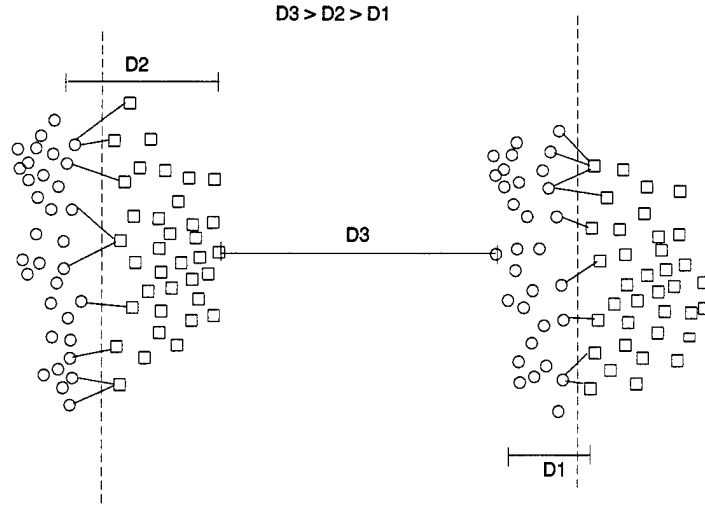


Figure 10 No Tomek links are generated in the middle region. This is because each opposing class is not close enough to the other across this divide.

paradigm are very similar to that of GRaCCE. That being said, the Park and Sklansky's PLC construction technique has some major shortcomings. The first is that a large number of Tomek links are generated if the classes in the data have a high degree of overlap. This can result in a large number of hyper-planes which overfit the data (100). While this condition can be mitigated by using an edited kNN procedure (150), this solution is not discussed by the authors. Perhaps the most fundamental problem, however, is that of completeness. In particular, the authors make no effort to ensure that the Tomek links generated are sufficient to represent the actual class boundaries; this is problematic as the links are generated only once, at the start of the algorithm. As Figure 10 shows, for some types of data sets, the distance metric can cause some boundaries to be ignored. As a result, the PLC algorithm is unable to generate sufficient Tomek links to completely enclose each cluster.

2.4 Evolutionary Methods

Up to this point, the rule induction methods described have been deterministic in nature or used random search in a limited manner. In this section, the application

Table 4 Piece-wise Linear Classifier Summary.

Characteristic	Description
Search Method	Uses deterministic search.
Search Criteria	Each partition separating two class must maximize the number of Tomek links (separating the same classes) it cuts.
Data Related Assumptions	Data can be discrete or continuous.
Rule Related Assumptions	Partitions must be linear.

of evolutionary algorithms (EAs) to the problems of pattern classification, decision tree induction and rule induction are discussed. Much of this material assumes a familiarity with genetic algorithms (GAs) and genetic programming (GP); accordingly, a short tutorial on these subjects is provided in Appendix A.

2.4.1 Genetic Decision Tree Algorithms. GAs are frequently employed as tuning mechanisms for other algorithms. In terms of the pattern classification problem, hybrid algorithms have been developed that use GAs to enhance the performance of existing DTI techniques. A typical method employed by these hybrids is to use GAs to preprocess the data set input to the induction algorithm. For example, DeJong et al (25) used GAs in conjunction with the ID3 algorithm (105) to improve decision tree quality. Each tree was constructed using the features selected in each individual's chromosome. The quality of the resulting tree in terms of accuracy and structural simplicity is then evaluated by an objective function. By mating individuals which resulted in the fittest trees, DeJong was able to evolve a superior feature set for DTI.

Turney (142) proposed a similar approach where the objective was to minimize the cost associated with classification errors. In contrast to DeJong, however, his GA searched for a set of biases for each attribute in the data set (making it more akin to feature extraction). The induction algorithm (C4.5) was modified to take

these biases into account when selecting attributes to partition. Another approach (developed by Janikow (66)) uses GAs to optimize the fuzzy value boundaries for domain values of a given attribute. For example, if the attribute label is *speed* and the domain values are *slow*, *medium*, and *fast*, then the GA is used to determine the actual *speed* thresholds for each domain value. By changing these thresholds, the structure of the tree induced (and its corresponding performance) can be streamlined. These approaches are appealing because they transform a greedy search into an evolutionary one while retaining proven induction algorithms.

2.4.2 Classifier Systems. Classifier systems (CSs) are one of the oldest evolutionary methods for machine learning. Generally speaking, CSs use GAs to evolve production rules that respond to and act on a given environment (refer to Table 5). According to DeJong (24), there are two basic classifier paradigms: the Michigan approach exemplified by Holland and Reitman's CS-1 system (60) and the Pittsburgh approach exemplified by Smith's LS-1 system (127). The main difference between these two stems from the composition of their populations. Systems using the Michigan approach maintain a population of *individual* rules while those using the Pittsburgh paradigm maintain a population of variable-length *rule sets*. In each case, members compete with each other for space and priority in the population. While they have been applied to a broad range of problems (41), this discussion centers on their use for learning classification rules from data. For more information on this topic, Gordon (46) provides a comprehensive reference.

Figure 11 depicts a block diagram of a rule induction CS based on the Michigan paradigm (78). Assuming a data set with d discrete features and m classes, each member of the population represents a fixed-size, rule of the form:

if <antecedent> then <consequent>, where
 <antecedent> := $\langle c_1 \rangle \cap \dots \cap \langle c_d \rangle$, and
 <consequent> := $\omega_1, \dots, \omega_m$.

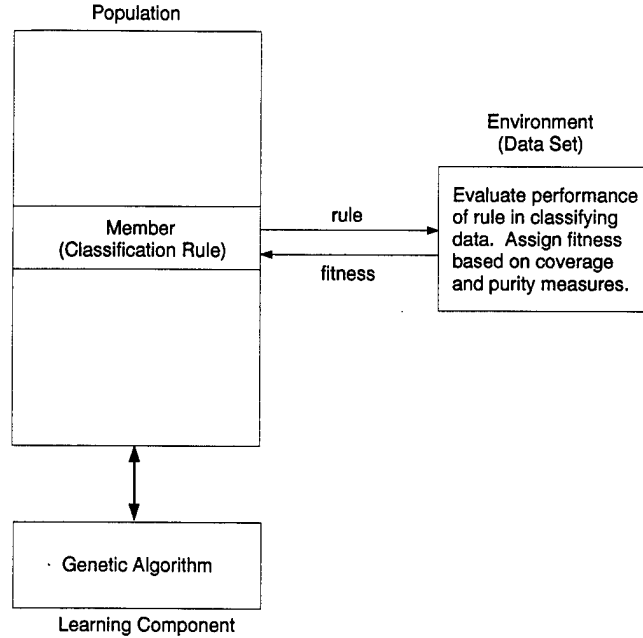


Figure 11 Block diagram representation of a GA-based Classifier for Rule Induction.

The antecedent of each rule is a conjunction of d conditions, each of which is a test on a unique feature. The consequent contains a label corresponding to one of the m classes. When a rule is applied to a given feature vector, the state (true or false) of each condition is determined by matching the value of each feature to the settings in its corresponding conditions. If the feature's value is consistent with these settings, then the condition evaluates to true; if not, the condition is false. When the antecedent evaluates the true, the class label specified in the consequent is assigned to the feature vector.

For this scheme to function, the GA chromosome structure (genotype) must be able to represent a given rule (phenotype). If a binary GA is utilized, then each condition must map to a series of genes, each of which represent an on/off for each discrete value of its associated feature. For example, if the i th feature has an enumerated data type of {red, white, blue, green}, then its associated condition is

represented by a 4 bit binary sequence. Thus, a sequence of 0101 represents the following symbolic condition:

$$< c_i > := \text{if } (F_i = \text{white or green}).$$

The fitness of a given rule is determined by its coverage and purity. *Coverage* is the percent of the total instances the antecedent matches. *Purity* is the class diversity of the covered feature vectors. Of course, rules with wide coverage and high purity are most fit. By using this evaluation scheme, the need to model the consequent in the chromosome is eliminated; this simultaneously simplifies the GA and the search space.

Several classifier systems have been specifically designed for learning rules from data. These systems are typically based on (or hybrids of) the Michigan or Pittsburgh paradigms. Examples of such systems include REGAL (96) and DeJong's GABL (24). REGAL is based on the Michigan approach, but differs in that it allows the rules in the population to overlap with each other. In contrast, GABL uses the Pittsburgh approach, with each member of the population representing a variable length rule set; the encoding for each rule under this scheme is similar to that described above. A commercial system, GA-MINER, is being developed based on work done at the University of Edinburgh (35). What distinguishes GA-MINER from its competition is the use of pattern templates to find *interesting* rules.

Because the evolved rules are specified in Disjunctive Normal Form (DNF), each rule is analogous to a path in a decision tree (extending from the root node to a leaf). Since rules produced are similar to those produced by ID3 (104), classifier systems suffer from the same disadvantage — the way the data is separated may be inconsistent with the natural structure of the data (refer to Figure 5). As a result, the rules produced might be overly complex and/or misleading. In addition, classifier systems often require numeric features to be categorized (mapped to a symbolic

Table 5 GA-Based Classifier System Summary.

Characteristic	Description
Search Method	Uses evolutionary search (GA-based).
Search Criteria	Maximizes rule coverage and purity.
Data Related Assump- tions	Data must be discrete.
Rule Related Assump- tions	Rules utilize univariate, linear partitions. Partitions are specified with regard to discrete categories. The two major approaches for rule evolution are the Michigan and Pittsburgh paradigms.

domain). As discussed in Section 3.5, the way the discrete categories are defined can significantly influence the complexity of the induced rule set (66).

2.4.3 Genetic Program-Based Classifiers. Koza (77) introduced the idea of using genetic programs to perform classification in his landmark book *Genetic Programming*; in it, he showed how GPs could be used to implement decision tree classifiers. Perhaps the first serious application of the GP paradigm to the problem of classification was accomplished by Tackett (134); the general outline of this approach is summarized in Table 6. The specific application was an Automatic Target Recognition (ATR) system to detect a variety of vehicles (including tanks, trucks, etc) in Infra-Red (IR) images.

To implement his classifier, Tackett utilized the output of the Hughes Multi-Target Acquisition Processor (MTAP) ATR system. MTAP generated a set of statistical features describing anomalies detected within the image. The GP's terminal set consisted of these features along with a random number generator. A very primitive function set was utilized containing the addition (+), subtraction (−), multiplication (×), protected division (%) and conditional (\leq) operators. Each GP's fitness is based on its false alarm rate (FAR) for a given probability of detection (PD). Objects detected by MTAP are classified by comparing the output of each GP's root node (ϕ) to a simple threshold. If $\phi > 0$ then the object is classified as a target;

otherwise, it is tagged as clutter. The classifiers evolved using this approach were compared against two other methods: Multi-Layer Perceptrons (MLPs) and Binary Decision Trees (BDTs). Tackett's results show the GP outperforming the MLPs and BDTs for high PD rates (96%).

Tackett's success in using GPs as classifiers spurred others to experiment further. Teller and Veloso (137) developed the *Parallel Architecture for Discovery and Orchestration* (PADO) to detect images and sounds in large databases. PADO uses a bi-level classification scheme that weights the ability of a pool of GPs to identify objects of a given class. Another innovation incorporated into PADO is the use of "smart" operators that learn the best way to evolve a GP over time. According to the authors, PADO's classification accuracy varies between 50% and 80% depending on the type of sound or image being retrieved. Their approach of using a pool of solutions (rather than a single one) to classify an object is utilized to some extent in GRaCCE.

Iba (64) developed another variant of the GP paradigm for classification (known as STROGANOFF). STROGANOFF differs from Tackett's approach in two basic ways. The first difference relates to the nature of the GP structure itself. Iba's GPs are BDTs in which nodes consist of quadratic polynomials and leaves are the input variables. Since no function set is used, the nodes are processed by solving each quadratic using a regression technique. As a result, the classifier consists of a hierarchical equation that must be solved through using recursive, multiple regression analysis. The second difference is the use of a minimum description length (MDL) based objective function for evaluating tree structures. This fitness metric trades off the structural complexity of the tree against its mean squared classification error (MSE). Iba demonstrated STROGANOFF's versatility by applying it to a number of pattern recognition problems (including a temporal problem). In these experiments, STROGANOFF outperformed traditional GPs in terms of accuracy and generalization ability. Perhaps the biggest disadvantage of STROGANOFF is its

Table 6 Genetic Program based Classifiers Summary.

Characteristic	Description
Search Method	Evolutionary search (GP).
Search Criteria	Maximize classifier accuracy and minimize complexity.
Data Related Assump- tions	None.
Rule Related Assump- tions	Primarily utilized for two-class pattern recognition problems. The evolved program must use the specified function and terminal sets. In addition, the GPs can produce non-linear class partitions.

computational complexity due to the multiple regression analysis that must be done for each candidate solution. In addition, the numerical nature of STROGANOFF's solutions make them difficult to interpret.

Sherrah (118) proposed using GPs to evolve features for input to simple classification algorithms (such as Maximum Likelihood or Perceptrons (10)). Using this approach, data is clustered using an unsupervised algorithm. A decision tree structure is then used to iteratively partition the data into individual classes. Each node in the decision tree has two components: a feature preprocessor and a classifier. The feature preprocessor is a GP that is used to fashion new features (out of existing ones) for input into the decision node's classifier. Sherrah dubs this architecture the *Evolutionary Self-Structuring Classifier* (ESC). A follow-up article by Sherrah (117) indicates marginal (but unimpressive) success when applying ESC to several benchmark problems.

Lastly, Konstam (76) uses a hybrid GP/GA approach to evolve classifiers for two-group classification problems. In particular, Konstam first uses GP to evolve the functional form of the discriminant function. He then employs a GA to evolve the coefficients that maximize the accuracy of the classifier. Although interesting, using two evolutionary paradigms to construct partitions make this approach extremely inefficient.

2.4.4 Mining the Genetic Program. Despite their success as classifiers, GPs have proven extremely difficult to interpret; this is a major obstacle to their use in data mining problems. One possible solution lies in identifying those subexpressions (or building blocks) in the GP that make it a good classifier. These subexpressions should be much easier to interpret than the GP as a whole. While a number of algorithms (such as Goldberg's Messy GAs (44)) were developed to locate good building blocks in GA chromosomes, almost no comparable work has been done for GPs.

The exception is the research accomplished by Tackett (135). Tackett sets up a framework by which to track the migration of unique subexpressions (traits) within a GP population during the evolution process. Specifically, he collects statistics for each subexpression (such as conditional fitness, peak fitness, frequency, time of creation, etc) to determine its relative worth. The theoretical basis of this approach is Holland's Schema theorem (61) which implies that traits that contribute to above average fitness will have a high frequency of occurrence in the population.

Although Tackett's research yielded some interesting findings, his technique is difficult to automate because it requires careful interpretation of these statistics over the course of a run. A more tractable approach for measuring saliency is to prune individual subexpressions from the GP and measure their direct impact on GP fitness. While Tackett mentions pruning as a possible alternative to his approach, he indicates the associated computational cost would be prohibitive. Preliminary experimentation (87) has shown that salient GP subexpressions routinely partition the data in effective, but non-intuitive ways (87). For example, the partitions evolved are frequently non-linear. Thus, despite this elaborate procedure, there is no guarantee that the end product will be easily understood. This is a fundamental reason why GPs were not actively pursued in the implementation of GRaCCE.

2.5 Association Rules

Association rules (ARs) are among the most basic type of rules that can be learned from data. Agrawal and Srikant (2) define a generalized AR as an implication of the form: $X \rightarrow Y$, where X and Y are sets of discrete conditions (such as $X = x_1$ and $Y = y_2$) in a database (D) such that $X \cap Y = \emptyset$. Since rules are rarely universal, metrics exist to describe the extent to which rules hold in D . The two principle metrics used in this regard are confidence and support. The rule $X \rightarrow Y$ holds in D with confidence c if $c\%$ of feature vectors where X holds also satisfy Y . The rule is also said to have a support s if the association $X \cap Y = \emptyset$ is true for $s\%$ of the feature vectors in D .

The algorithms for mining ARs from data are fairly straightforward. The three basic phases of AR mining are described in (2) and summarized below:

1. Find all sets of items whose support is greater than a user specified threshold. The sets of such items are called frequent item sets. When identifying item sets, emphasis is put on finding the largest possible item sets which meet the support threshold.
2. Use the frequent item sets to generate the desired rules. This is accomplished by breaking up item sets into separate components and establishing relationships between them. For example, if (X, Y, Z) is a frequent item, then any of its subsets is also a frequent item. We can then derive the rule $XY \rightarrow Z$ if this relation meets the minimum support and confidence thresholds set by the user. Han and Fu (50) suggest using the hierarchical concept relationships to make substitutions in association rules in order to increase their support, thereby improving the quality of the derived rules. Note that generation of redundant rules (such as $X \rightarrow Z$) are avoided with this method.
3. The next step is to prune the set of rules discovered. Though AR algorithms tend to discover many rules, only a few of these will be deemed "interesting."

To prevent the user from being overwhelmed by trivia, it is often desirable to discard all rules that are not determined to be useful.

While association rules are easy to generate, their usefulness is often dubious. This is due to two main factors. First, AR algorithms produce rules based on *any* type of association in the data. Unlike the induction algorithms discussed thus far, associations may be generated for any combination of features; class labels need not be part of an AR. This arrangement leads to rule sets with a high degree of redundancy. As a result, disjoint rules are presented to the user without a coherent framework. While simple methods can be used to minimize the confusion (such as sorting rules by consequent attribute), it is often incumbent on the user to bring order out of chaos. Second, the methods most widely used for pruning rules are domain independent. By using generic criteria for pruning, these approaches run the risk of destroying information that is potentially useful to the end user. A more thorough discussion of techniques for identifying interesting rules can be found in Chapter III.

2.6 Induction of Bayesian Networks

Up to now our discussion has focused on the induction of decision rules which result in the assignment of an absolute class label. In reality, however, classification decisions are frequently made with a degree of *uncertainty*. For example, in a two class problem, a given feature vector may have a 70% probability of belonging to the first class and a 30% probability of belonging to the second. From a decision making point of view, rules that yield class probabilities could be desirable. A treatment of the probability theory underlying the class selection process is given in Section 5.1.

Bayesian Networks (BNs) provide a means of obtaining this information. BNs are directed acyclic graphs (DAGs), where the nodes are random variables (RVs) and the arcs specify the independent assumptions that are held between these variables. Thus, the BN represents the joint probability distribution over all the RVs in its

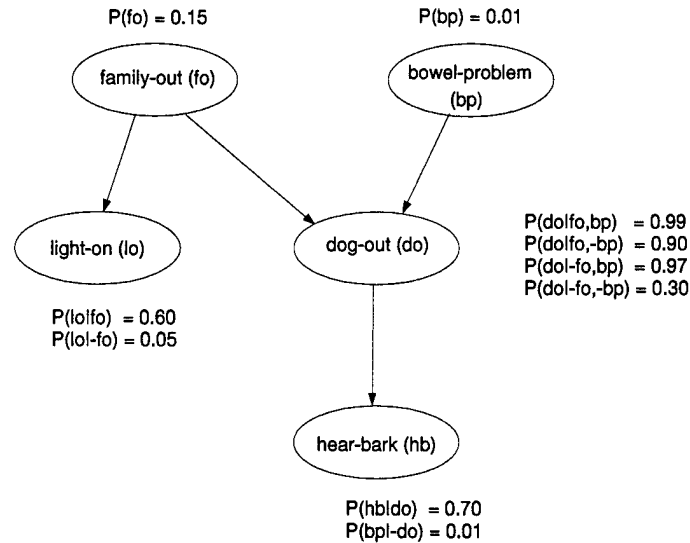


Figure 12 Bayesian Network for Charniak's "Family Out" problem.

domain. Once constructed, a BN enables the conditional probabilities of the nodes in the network to be computed given evidence (i.e., the values of those nodes which have been observed). As a result, BNs are a powerful tool for both representing uncertainty and performing probabilistic inference³. The BN for Charniak's classic "Family Out" problem (15) is shown in Figure 12. In this example, the "family-out" and "bowel-problem" nodes are equivalent to classes since only they have prior probabilities.

The task of constructing a BN can be separated into two subtasks: *structure learning* and *parameter learning* (79). Structure learning involves identifying the topology of the network. This requires knowledge of the causal relationship between RVs. In the Figure 12 example, the "dog-out" RV causes "hear-bark." The parameter learning subtask involves estimating the numerical parameters for a given network topology. Of these two tasks, establishing a viable structure for a BN is by far the more difficult. A summary of the BN construction issues is given in Table 7.

³Inference is based on Bayesian decision theory. Please refer to Chapter V for a short tutorial on this subject.

A frequently used procedure for BN induction is the K2 algorithm developed by Cooper and Herskovits (19). Given a database D and an approximate causal ordering for the RVs (B_s), K2 uses a greedy heuristic to induce a network that maximizes their joint probability, $P(D, B_s)$. At its conclusion, K2 returns an estimate of how closely the induced BN fits the data. The complicating factor is that constructing the optimal BN is predicated on specifying the “correct” causal ordering. A number of techniques have been developed to help solve this hard, combinatorial problem. Verma and Pearl (146) proposed an algorithm for establishing causal order between RVs (if one exists at all). In addition, Larrañaga (79) used a GA-based method to search for an optimal ordering. Singh (122) provides a survey of on-going research efforts in this area.

While BNs have garnered some interest as a data mining technique, it makes a number of very questionable assumptions. Perhaps the most dangerous assumption is that there exists a *causal relationship* between the recorded variables in the database. In many cases, the presence of latent variables can give the illusion of causality. A latent variable is any unrecorded feature that varies among recorded units and whose variation influences recorded features. The result is an association among recorded features not in fact due to any causal influence of the recorded features themselves (40). The classic example of the presence of a latent variable is a correlation that shows the crime rate of a given community rising with the number of churches. While such an association may exist, it would be incorrect to assume that one variable has a causal relationship to the other. Since automatically generated BNs are built from the recorded data, the potential for misleading causal inferences is high.

2.7 Rule Extraction

Rule extraction is an indirect form of rule induction. Specifically, instead of learning classification rules directly from the data, this method extracts rules from a

Table 7 Bayesian Network Summary.

Characteristic	Description
Search Method	Open (can be deterministic or evolutionary).
Search Criteria	Given a database (D) and a causal ordering of random variables (B_s), maximize $P(D, B_s)$.
Data Related Assumptions	Data must be discrete. Distinction between class and data can be blurred.
Rule Related Assumptions	Rules are probabilistic in nature. The form of the generated rule set is heavily dependent on B_s .

classifier already trained on the data. Once trained, a classifier provides a mapping from a set of input variables x_1, \dots, x_d to an output variable y whose value represents the class label $\omega_1, \dots, \omega_m$ (10). Thus the mappings embedded in the classifier structure are an approximation of the rules it uses to classify data. The difficulty of extracting these rules is heavily dependent on the characteristics of the target classifier.

A textbook example of the type of algorithm described above is the NeuroRule system (84). NeuroRule mines rules from trained Multi-layer Perceptrons (MLPs), a type of neural network⁴. In general terms, MLPs are statistical processors that approximate the underlying distribution of the data. Figure 13 shows a three layer MLP network, consisting of an input layer, a hidden layer, and an output layer. The network structure is a system of nodes (activation functions) connected by weighted connections. During training, the MLP learns the appropriate weights in order to minimize the output error for a given set of inputs (i.e., the data set). MLPs have proven to be very accurate classifiers due to their ability to approximate non-linear class boundaries. Unfortunately, the classification rules learned by the MLP are encoded into the structure of the graph and the weights that link the nodes. Since these are numerical in nature, articulating the rules represented by the MLP structure is a difficult problem.

⁴Macy and Pandya (86) provide a comprehensive reference on neural networks.

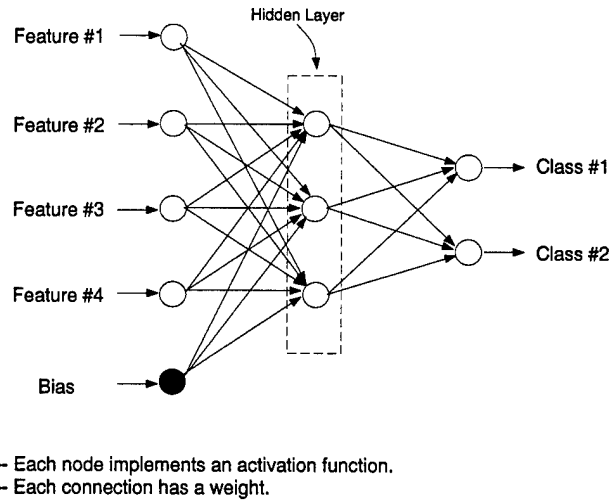


Figure 13 A simple, fully connected Multi-layer Perceptron (3 layer, 4 input, 2 output).

Once an MLP has been successfully trained, NeuroRule prunes the network to reduce its complexity while maintaining a minimum level of accuracy. Simplifying the MLP is important because it reduces the possibility of extracting rules that are overly complex, redundant or extraneous. Pruning is an iterative process which involves removing connections whose weights fall below a given threshold. After each pruning cycle, the MLP is retrained to determine if it still meets its minimum accuracy. If it does, the threshold weight threshold is incrementally raised; if not, the network from the previous cycle is restored and the pruning process is terminated.

After pruning is completed, classification rules are extracted from the simplified MLP. The activation thresholds of each hidden node in the network are recorded (through testing) and clustered into discrete categories. A table is then constructed in which all activation states for the set of hidden nodes is represented; the corresponding network output value is also computed for each state. Sets of activation values (SAVs) that produce a given output state are then derived. Through substitution, these SAVs can be used to identify the input value ranges that produce the corresponding outputs states. These input range - output state combinations can be structured in IF/THEN form to provide the final set of rules. In practice, the rules

yielded by NeuroRule are similar in form to those generated by C4.5 (105). Indeed, the authors (Lu, Setiono and Liu) report producing smaller rule sets than C4.5, but with a comparable level of accuracy (85). The principal disadvantage of NeuroRule, however, is its lack of efficiency with respect to C4.5.

In contrast to NeuroRule's top down approach, Rathbun (106) developed a unique bottom-up approach to MLP design, known as the MLP iterative construction algorithm (MICA). A distinguishing characteristic of MICA is the serial training of hyper-planes to separate the classes using the Ho-Kaphyap method (29). Additional hyper-planes are added until a classification accuracy of 100% is achieved on the training data. While Rathbun's intent is to use this technique to construct MLPs, the hyper-planes can also be utilized for decision rule set construction. The drawback here is that the 100% accuracy requirement may result in the generation of many hyper-planes that are unrepresentative of the natural class boundaries.

A similar technique for constructing radial basis functions (RBFs) was developed by Wilson (151). The RBF iterative construction algorithm (RICA) uses the Shapiro-Wilk metric (116) to determine how well a given data cluster fits a Gaussian distribution (129). If the fit is good, then the cluster is approximated with a single RBF; if not, it is split up into separate (more Gaussian) clusters. This technique iterates until all clusters are assigned an RBF. Once a set of satisfactory RBFs are identified, a geometric structure technique, such as a Gabriel Graph (65), can be employed to generate hyper-planes to separate neighboring RBFs of differing class. Of course, the effectiveness of this technique is predicated on assumption that the data has a Gaussian distribution (which is often not the case).

2.8 *Summary*

While a wide spectrum of important decision rule induction algorithms were covered in this chapter, it is far from a complete survey of the field. Nonetheless, a

number of valuable insights have been gained into why induction algorithms can fail to yield optimal rule sets. These factors include:

- The NP-complete nature of the problem.
- The greedy nature of the induction algorithm.
- Simplifying assumptions about the data (i.e., variables are discrete).
- Simplifying assumptions regarding the rule structure (e.g., conditions are equivalent to univariate, linear partitions).
- Partitioning criteria which ignore the natural class boundaries.
- A tendency to over-fit the data.

Although some of these factors cannot be compensated for (i.e., the NP complete nature of the problem), most represent opportunities for improvement through better algorithm design. Consequently, specific strategies to rectify some of these problems are incorporated in the GRaCCE algorithm (discussed in Chapter IV).

III. Preliminary Issues

The focus of this research is mining compact decision rules from data. Just as the difficulty of mining minerals is dependent on the geology of the target site, the rule induction process is likewise influenced by the characteristics of the data set being mined. This simple analogy motivates us to examine how these characteristics impact the data mining process. The information presented in this chapter serves as a primer on these issues for the technical approach described in subsequent chapters.

3.1 Effects of Dimensionality

The features within a data set provide the information that make class discrimination possible. Because of this, it is natural to assume that increasing the number of features can always improve class discrimination. In practice, however, adding additional features can actually lead to a *degradation* in classifier performance (10).

Consider the problem of constructing a classifier for a data set with d features and M divisions per feature. Under these conditions, the classifier must learn M^d mappings to be effective. For a data set of finite size, fewer and fewer instances can be assigned to each mapping as the number of features increases. Eventually, a point is reached for every problem where insufficient instances exist to learn a given mapping. Bellman (8) termed this phenomenon the *curse of dimensionality* and stated that as the number of input features increases, the number of feature vectors must increase exponentially for accurate classification.

Another consequence of high dimensionality is the difficulty of accurate parameter estimation. For example, when using a Gaussian maximum likelihood classifier (138), the mean and covariance matrix of each class are usually not known and must be estimated from the training samples. For d dimensional data, the sample covariance matrix is singular, and therefore unusable if fewer than $d + 1$ samples are available from each class (29; 58). This is a problem for induction algorithms

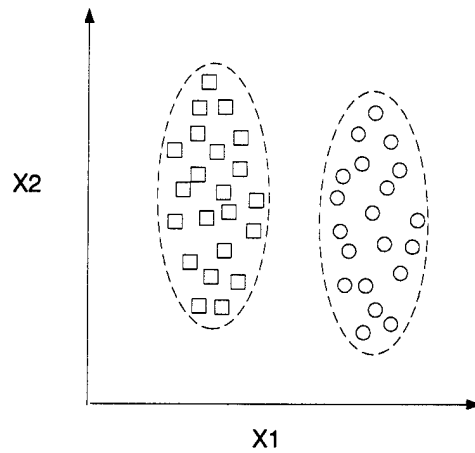


Figure 14 Comparative Feature Saliency.

(such as decision trees) which partition the data into smaller and smaller subsets; eventually the number of samples in a subset becomes sufficiently small that the partitions under-fit the data.

The above problems are often compounded by the fact that not all features are created equal. The usefulness of a feature in discriminating between classes is known as its *saliency*. In any given data set, some features may be good discriminators (high saliency), while others may not (low saliency). In the example shown in Figure 14, feature $X1$ is highly salient while feature $X2$ has almost none. Retaining features with low saliency is detrimental because it increases data requirements while contributing little to the classification mapping function. Even within the subset of features with excellent saliency, there may exist features that are highly correlated with each other. Maintaining redundant features is of marginal utility to the classification process.

3.2 Feature Selection

One of the surest ways to banish the curse of dimensionality is to reduce the size of the feature set. The basic problem is this: given a data set containing d features, how do we select the best subset of m features for performing classification

where $m < d$? Exhaustive search is not an option since finding the best m features for a problem of any reasonable size involves the evaluation of a prohibitive number of possibilities. For example, selecting the best 10 out of a set of 20 requires the evaluation of $\binom{20}{10}$ or 184,756 possibilities. As a result, more tractable approaches for feature selection are necessary.

One approach is to evaluate the *intrinsic* saliency of a given feature. Perhaps the most basic method for accomplishing this is the Fisher Discriminant (34). The Fisher Discriminant for a two class problem is defined as

$$f = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \quad (10)$$

This technique rates the saliency of a feature based on how far each of the class means are separated relative to the spread of their variances. Thus, the larger the f value, the better the class separation and, therefore, the more salient the feature. A generalized F-ratio (for multi-class problems) is given by

$$F = \frac{\text{Variance of the means (over all classes)}}{\text{Mean of the variances (within classes)}} \quad (11)$$

Additional saliency measures that expand on the F-ratio concept have been developed by Fukunaga (37).

While it is computationally efficient to measure intrinsic saliency, these approaches have three fundamental disadvantages. First, the data are assumed to conform to a certain distribution (Fisher assumes a Gaussian distribution); of course, the results suffer if this assumption is incorrect. Second, each feature is evaluated in isolation; as a result, the effect of feature combinations are ignored. As mentioned

earlier, if two features are both highly salient and correlated, then little new information is provided by selecting both of them. Finally, because these approaches are classifier independent, they may fail to account for differences in the way specific classification algorithms process individual features.

The last disadvantage has led to *classifier-dependent* approaches which select feature subsets based on performance on specific classifier algorithms. These methods evaluate a candidate subset by training a target classifier using only the data contained in the selected features. The winning subset is chosen based on criteria that optimize classifier accuracy while minimizing the number of features. Because the classifier must be trained for each candidate subset, it is much less efficient than the intrinsic methods discussed previously. Accordingly, classifier-dependent approaches have typically utilized classical search techniques such as forward search and branch and bound (17). While these methods have the advantage of bounding the search, they are not guaranteed to produce an optimal solution. In addition, several researchers (103; 124; 153) have demonstrated the effectiveness of GA-based stochastic search for feature selection.

3.3 Feature Extraction

Another strategy for minimizing the effects of dimensionality is to construct a new, more effective, feature set from the existing one. The premise here is that there exists a transform T to convert the original feature set (X) into another (X') which has lower dimensionality and is more conducive to classification,

One of the most fundamental feature extraction techniques is the *Karhunen-Loève (KL) transformation* (also known as *principal component analysis*) (10; 29). The KL transform calculates the covariance matrix of the data set and finds its corresponding eigenvalues and eigenvectors. Each of the eigenvectors (u_i) is a transform that acts on the original feature set to create a new feature; they are called principal components. The eigenvalue (λ_i) corresponding to each eigenvector denotes the

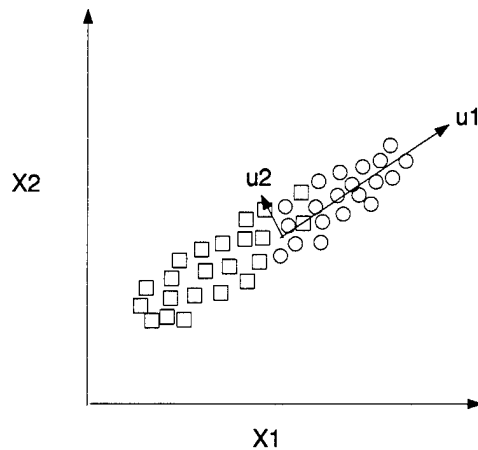


Figure 15 Karhunen-Loève Transform - Original two dimensional feature set is projected onto eigenvector u_1 , which has the greatest eigenvalue.

variance of the data over each principal component. The number of significant eigenvalues is taken as a measure of intrinsic dimensionality. The underlying assumption here is that principal components with the largest variances make the best features. Thus, the KL transform utilizes the eigenvalues as the basis for selecting the best m features. Thus, the chosen eigenvectors are used to transform the original data set to a new one. This concept is illustrated by Figure 15 where the two dimensional data set is reduced to a single feature, representing the projection of data along eigenvector u_1 . Of course, as Figure 14 illustrates, there is no guarantee that features with large eigenvalues are necessarily good class discriminators.

A more sophisticated feature extraction technique, known as Decision Boundary Feature Analysis (DBFA), was developed by Lee and Landgrebe (80). This approach winnows the data set using a k Nearest Neighbor (kNN) algorithm to remove data points that obscure the true class boundaries. With the classes more clearly separated, the algorithm proceeds to identify all potential decision boundaries. A vector normal to each decision boundary is then computed. As Figure 16 illustrates, each of these vectors represent the direction of optimal class separation for its respective decision boundary. The covariance matrix describing this set of vectors is termed the Effective Decision Boundary Feature Matrix (EDBFM). The

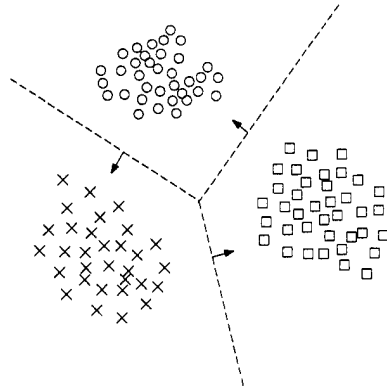


Figure 16 The EDBFM is the covariance matrix of unit vectors normal to the class decision boundaries.

EDBFM is used in a manner analogous to the covariance matrix in the KL transform to derive a new feature set. The key difference is that the eigenvalues of the EDBFM are far more accurate indicators of features that increase class separation.

Vafaie and DeJong (145) and Sherrah (117) have also demonstrated GA and GP-based approaches, respectively, for evolving new feature sets. Despite the effectiveness of these techniques, feature extraction is often undesirable in data mining applications. This is due to the extreme difficulty of interpreting the meaning of these new features in the context of the original domain. As a result, any rule using these features tends to be incomprehensible to the end user. Since a fundamental goal of this research is to produce an understandable rule set, feature selection remains the preferred approach.

3.4 *Processing Very Large Databases*

In addition to feature dimensionality, a data set can also be large with respect to the number of instances it contains. Indeed, the use of tera-byte databases, containing millions of instances, is becoming ever more prevalent in commercial applications (47). Processing these very large databases (VLDBs) poses a challenge for the DRI methods discussed in the previous chapter for several reasons. First, these methods are not linearly scalable with respect to the number of instances. Second,

the DRI algorithms require that all or a portion of the data set remain permanently in memory. This requirement causes these algorithms to execute far more slowly once the data set size exceeds the available memory (due to page swapping between disk and memory). Taken together, these factors limit the suitability of these algorithms for mining VLDBs. A common solution to these problems is to process a subsample of the VLDB (125). The principle disadvantage of this approach, however, is a loss of classification accuracy versus using all the data (14).

An alternative solution which addresses several issues in building a fast scalable classifier is the SLIQ decision tree algorithm (91). SLIQ handles disk-resident data that is too large to fit in memory. Rather than resorting to sub-sampling, SLIQ builds a single decision tree using the *entire* training set. SLIQ accomplishes this by utilizing a memory-resident data structure (called the *class list*) which is a fraction of the size of the actual data set. Even so, this structure can also overflow memory given a data set of sufficient size. The SPRINT system (114) (the successor to SLIQ), further extends these database size limits by distributing *both* the processing and data loads over N processors. Because SPRINT has proven to be extremely fast and scalable, other researchers have developed variations on this system which further enhance performance; these include the ScalParC (68), pCLOUDS (131), and BASIC (154) systems.

Another issue linked to the processing of VLDBs concerns the multi-dimensional, non-homogeneous structure of relational databases (RDBs). Treatment of this subject is particularly apt because the VLDBs used in modern applications are almost exclusively RDBs. Unlike the flat file formats processed by the DRI techniques discussed thus far, RDBs are organized in terms of composite data types known as *records*. Each record serves as a descriptor for some type of object (i.e., person, company, etc). A given RDB typically contains a multitude of different record types; records of like type are grouped together in *base table* structures. Different

record types are related through specific attributes (or keys) in their structure. A comprehensive tutorial on the organization of RDBs is provided by Date (23).

The incongruity between the flat-file format needed by the DRI algorithm and the structure of the RDB must be bridged in some manner. Given the diversity of data in an RDB, it is unrealistic to assume that data can be mined without detailed models (known as *catalogs*) of the RDB architecture and the target domain (i.e., how is one type of record related to another?). At a minimum, these catalogs are needed to navigate the RDB in order to extract and format data for processing. Thus, as part of preprocessing, structured query language (SQL) queries must be performed to extract the relevant subset of data from the RDB. This data (which can come from multiple base tables) is organized into a single, *derived table*. These derived tables have the flat file format required for processing by the chosen DRI algorithm. Because RDBs have the potential to yield a large, diverse set of derived tables, mining RDBs can be a time consuming, highly iterative process.

3.5 Discrete Features

A wide variety of Machine Learning (ML) algorithms (such as Classifier Systems (41)) exclusively operate on (or best with) symbolic domains. Features in such domains are discrete; that is, they hold values which correspond to fixed categories. For example, a discrete data type for weight could be {thin, normal, heavy, fat}. When ML algorithms are applied to data mining problems, the data sets processed often contain numeric features. In these cases, a discrete type must be created for each feature, along with a mapping from the numeric to the corresponding discrete values. Returning to the weight example, a mapping from numeric to discrete values could be defined as: {thin = (90 – 140 lbs), normal = (141 – 200 lbs), heavy = (201 – 250 lbs), fat = (250+ lbs)}.

The simplicity of the above example belies the difficulty of finding a discrete representation for a feature that is optimal. Perhaps the most basic approach is to

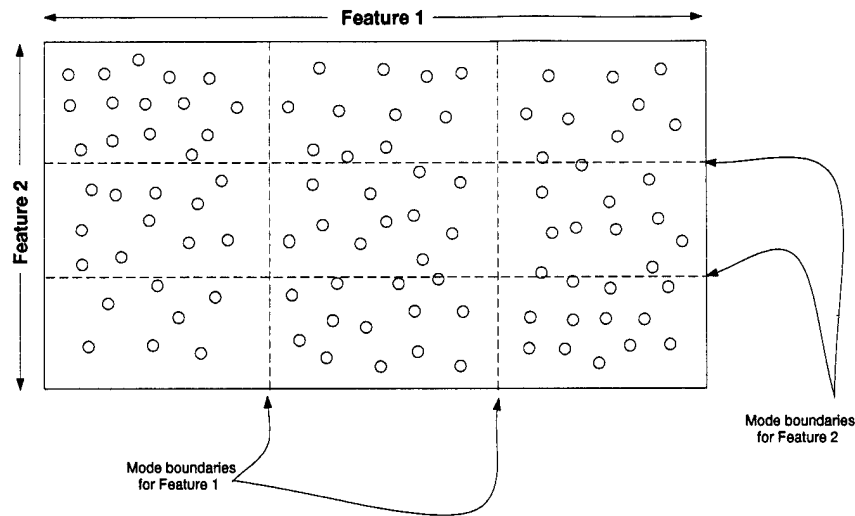


Figure 17 Discrete mode categories created without respect to class.

categorize based on the number of clusters (modes) in the feature's data distribution. Consider the two features depicted in Figure 17. Feature 1 has three very distinct modes which can be easily mapped to discrete categories. In contrast, Feature 2 has a fairly uniform distribution; as a result, category definitions for this feature are relatively arbitrary. Another complicating factor is the epistatic relationships between features and class. This point is illustrated by Figure 18. Observe that while Feature 1 has three distinct modes, choosing this configuration degrades classification accuracy. In contrast, accuracy is optimized when Feature 2 is divided into two categories; additional categories only result in unnecessary complexity. Also note that these observations are only possible when both features are analyzed together. Thus, if multiple features are necessary for class discrimination, the interaction of their category definitions can have a significant impact on classification accuracy and/or rule set complexity. For further reading on this subject, Fayyad and Irani (33) analyze different approaches to discretizing continuous-valued attributes for decision tree algorithms.

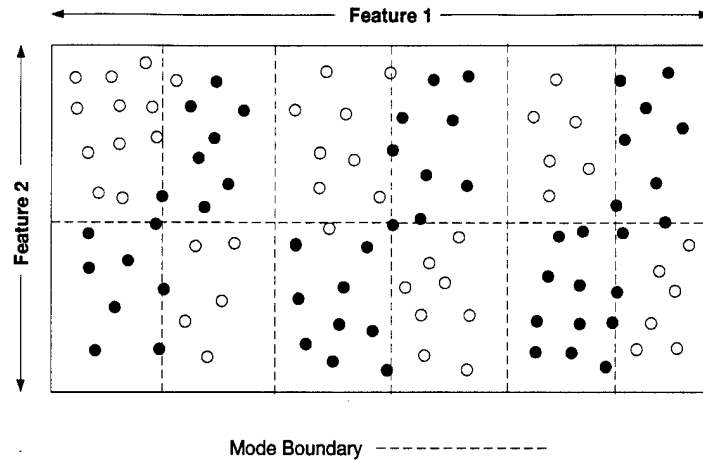


Figure 18 Discrete mode categories created with knowledge of class labels.

3.6 Modality of Data

Most data mining algorithms make some structural assumptions about the data. In situations where there is little a priori knowledge of the data, such assumptions (particularly relating to parametric distributions) may lead to poor or meaningless results (29). This is especially true for data sets with multi-modal densities. For rule induction problems, a class is multi-modal if it can be segregated into two or more natural clusters. The classic example of multi-modal data is the XOR data set shown in Figure 19. Multi-modal data sets have frequently proven problematic for greedy rule induction algorithms. This problem was summarized by Murthy (94) who noted: “the source of this difficulty is that the only information available to the goodness measures used is the distribution of object classes across the splits. However, building the ideal tree requires knowing that there are well defined *homogeneous clusters* in the attribute space. Existing decision tree methods cannot use any such structure information.” Given this critique, it would appear that the ideal rule induction algorithm should minimize assumptions and maximize adaptability with regard to the structure of the target data.

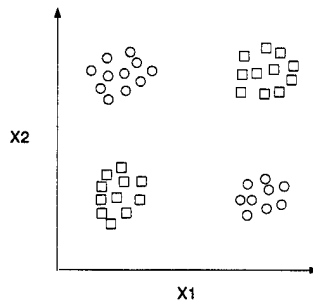


Figure 19 XOR Data Set.

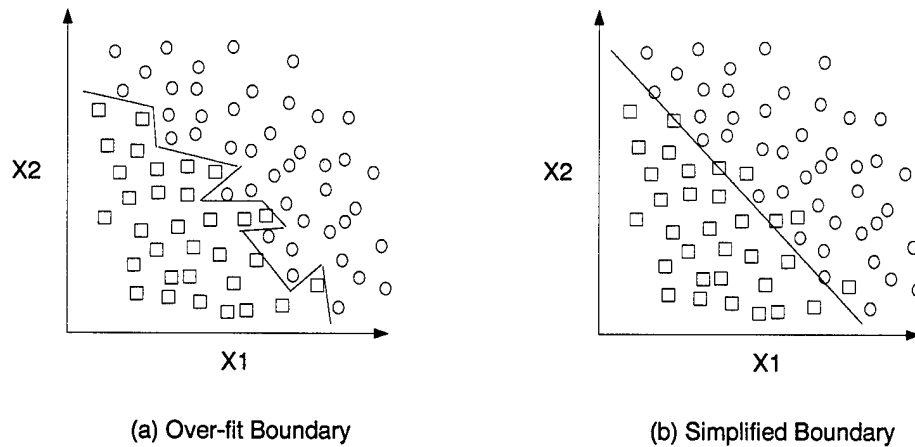


Figure 20 Over-fit class boundaries (left) tend to be more complex than those that aren't (right).

3.7 *Over-fitting and Training Set Composition*

Classifiers have a well established tendency to conform too closely to the patterns in the training data. Such over-fitting results in poor generalization, characterized by extremely good accuracy on the training data, but degenerate performance on the test set. In addition, solutions that over-fit the data tend to be more complex than those that do not (51); Figure 20 depicts such a situation. These problems make it obvious that over-fitting is a condition to be avoided.

The above discussion prompts the question: can anything be done to preempt (or minimize the effect of) over-fitting? ML algorithms employ a variety of mechanisms to control this condition. A technique known as early stopping (10) is used

to prematurely halt neural network training at the first sign of over-fitting. When evolving GP-based classifiers, a complexity metric can be added to the objective function to restrain the size of the GP (134). Over-trained classifiers can also be repaired. For example, Quinlan showed that decision tree pruning could decrease tree size while improving tree accuracy (105). A comprehensive treatment of this subject is also provided by Hand (51).

The composition of the training data set also impacts over-fitting. It has long been an article of faith in pattern recognition that more training data equals better performance (10; 29; 37). This belief is consistent with statistical methods (such as the Maximum Likelihood (59)) where more data allows more accurate parameter estimation. Yet, there is a growing body of evidence that suggests the contrary. Indeed, several researchers report that a reduction in the number of training instances results in a smaller rule set with equivalent accuracy (12; 67). Likewise, it has also been shown that adding training data *linearly increases* rule set complexity without an appreciable increase in accuracy (97; 98). These findings were validated on several different rule induction algorithms (18; 105).

Although research suggests that thinning the data set prior to training the classifier can help preempt over-fitting, it does not imply that winnowing can be done blindly. A significant factor affecting classifier accuracy is the composition of the training set. A number of researchers have found that different subsets of a given data set can yield substantially different error rates for the same classification method (108). This variance in error increases as the size of the subset decreases (11; 75) and/or the dimensionality of the data is high (28). Clearly, classifier accuracy can suffer if the training set is too large or too small.

A balance between these two extremes can be achieved by eliminating data that contribute little to classification accuracy. A large body of work exists to support this contention. Wilson (150) shows that the edited Nearest Neighbor technique resulted in error reduction. Devijver and Kittler (27) took this one step further by proving

this procedure is asymptotically Bayes-optimal with respect to the original data set. Experiments by Oates and Jensen (97) show that removing misclassified instances from the data set outperforms random winnowing of the training data. In addition, Aha (3) demonstrated that filtering instances based on records of their contribution to accuracy improves the accuracy of the resulting classifier. This strategy has the advantage of denying the classifier the *opportunity* to learn class outliers (data on the wrong side of the Bayes decision boundary). Such data is often easily misclassified and, therefore, is of questionable value; mislabeled samples may also fall into this category (12). The *marginal* reduction in training set size achieved by removing such noisy data shows promise as a way to provide an effective shield against over-fitting.

3.8 Missing Data

Real-world data is rarely perfect. In particular, many off-the-shelf data sets contain instances where attribute values are missing from the data. Processing such data sets is a challenge for any data mining algorithm. Many researchers have addressed the problem of compensating for missing data; a few of these solutions are summarized here. The simplest approach is to merely remove instances with missing data (36). Although effective for situations where a wealth of data exists, this approach may lead to serious biases in the presence of large amounts of missing data (83). In addition, it may not be practical when the amount of data available is fairly small. In the latter case, it is essential to make full use of the information available in the incomplete samples.

If we want to retain the missing instances, we must select a strategy for *replacing* the missing attributes. Simple replacement strategies include substituting a default value (i.e., 0). This naive approach can alter the natural cluster structure of the data by creating additional modes (39). For data mining applications, this could lead to the induction of non-existent rules. Little and Rubin (83) describe approaches that estimate the missing value from the mean of the observed values

for the missing attribute (mean imputation) or by sampling from the unconditional distribution estimated from the observed values (hot-deck imputation). While easily implemented, each of these methods is biased because they treat the replacement value as the actual missing value (121).

Perhaps the best techniques are those that develop integrated models for the missing attributes and base inferences on the likelihood given by those models (10). One such technique is the Expectation Maximization (EM) algorithm developed by Dempster (26). This algorithm iteratively develops a model of data composed of a mixture of different Gaussian distributions. Ghahramani and Jordan (38) propose updating the missing data in conjunction with the EM algorithm. In particular, the missing data is re-estimated every time the distributions are changed. Updating continues until the estimated error stabilizes.

Another option is a Monte-Carlo type simulation technique called multiple imputation (113). This method generates $m > 1$ simulated values for each missing datum. The resulting m versions of the data are then analyzed by standard complete-data methods and integrated into a single model (such as a Bayesian network (121)) which captures uncertainty information about the potential replacement values. From a practical standpoint, however, it is unrealistic for non-Bayesian rule induction algorithms to maintain multiple versions of the data; a single replacement value for each missing datum must be chosen. Unfortunately, such a change reintroduces bias, effectively negating the primary advantage of these computationally expensive approaches.

3.9 Normalization

One of the most common forms of input preparation is normalization of data. Normalizing an attribute scales its values so that they conform to a normal distribution with zero mean and unity variance, denoted by $N(0, 1)$. Equation 12 converts

an the i th attribute of a database (X_i) to its normalized form, X_i^* for the j th feature vector.

$$X_i^*(j) = \frac{X_i(j) - \mu_i}{\sigma_i} \quad (12)$$

The advantage of normalization is that it filters out the wide variations in values caused by differences in data types. For example, in a given feature vector, one attribute X may range in value from 0 to 1, while attribute Y may range in value from -1000 to $+1000$. In a k Nearest Neighbor (kNN) algorithm, normalizing the data prevents one attribute from unfairly biasing the Euclidean distance calculation between different feature vectors. With respect to neural networks, normalization ensures that the network weights can be initialized (and updated) in a similar manner. If normalization were not performed, sufficiently large inputs would saturate the network, making its output unreliable.

Despite its advantages, normalization is not always desirable. Consider the data in Figure 21. Before normalization, the two classes are linearly separable; afterwards, they are not. From a data mining perspective, when interpreting decision rules generated from normalized data, it is crucial to take into account how each attribute is scaled. This requirement can add significantly to the difficulty of this task since the normal distribution is a non-linear function. Given this, the decision to normalize data should be an informed one.

3.10 *Identifying Interesting Information*

While the current generation of data mining algorithms excels at finding patterns in data, in many ways this is half the battle. One of the fundamental problems with many data mining tools is that they can yield a glut of findings, most of which are of no interest to the user (101). Thus, it is often necessary for a human to

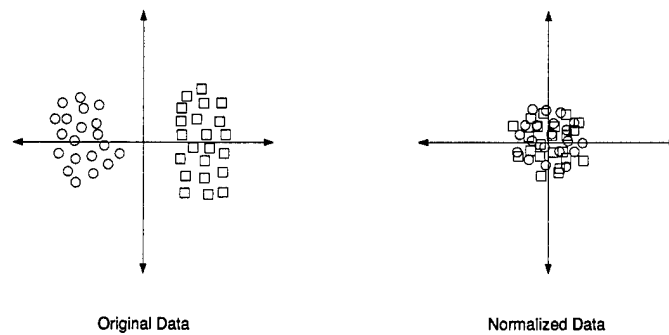


Figure 21 Normalization can sometimes impair separability.

sift through the mined information. Needless to say, this is long and tedious work. Consequently, a number of researchers have been working on metrics which rate the interestingness of the discovered patterns. The end goal of this research is to produce systems which can automatically mine information that is of vital interest to the end user.

Research in this area falls into two main categories: subjective and objective. Subjective techniques use information (i.e., a knowledge base) about the domain described by the database to determine the interestingness of a given pattern. For example, Silberschatz and Tuzhilin (120) tag a pattern as interesting if it is either unexpected or actionable. An unexpected pattern is contrary to the firm beliefs of the user (represented by using a Bayesian network). Actionable patterns are those that impact conditions that the user can change. Along a similar vein, Piatetsky-Shapiro (102) identifies interesting patterns based on deviations between an observed value of a measure (extracted from the database) and a reference value. This technique requires the user to specify a domain specific reference value that is used as a basis for comparison. In addition, thresholds must be specified to identify when a deviation becomes significant. The use of deviations is a simple way to identify things that differ from our expectations; the author argues that it is this quality that makes information interesting.

Another subjective approach (73) uses predefined templates to indicate relationships of interest to the user. These templates are similar in nature to Structured Query Language (SQL) queries; results (in the form of association rules) matching these templates are reported back to the user. Lastly, Han and Fu (50) use differences in expected confidence and support based on an association rule's position in its concept hierarchy to assess interestingness. For example, large support is more likely to exist at higher levels in the concept hierarchy (such as milk and bread) rather than at lower ones (such as a particular brand of the above). Any association rules that deviate from this expected pattern is considered interesting. Such subjective approaches best reflect the user's preferences when the underlying models are comprehensive, complete and accurate. Of course, insuring this requires a great deal of preparation, even for relatively simple domains. As a result, subjective metrics are not suitable for general purpose rule induction techniques.

An alternative to these labor-intensive approaches is to measure interestingness based entirely on statistical regularities in the data. These types of objective metrics are appealing because they are domain independent. Chan and Au (13) developed one such measure, called *adjusted difference*. Given an association rule $X \rightarrow Y$, this metric determines if the $P(X|Y)$ is significantly different from $P(X)$. If this condition is true, then the association between X and Y is deemed interesting. The most commonly used objective measurements, however, are the confidence and support metrics discussed in Section 2.5. For supervised classification problems, decision rules of most interest are those that isolate large class homogeneous regions within the data. Such rules are characterized by high levels of support (coverage) and confidence (purity).

3.11 Summary

This chapter addresses how the characteristics of the underlying data set can impact the data mining process. In addition to understanding the problem, however,

strategies are introduced that can improve the processing of difficult data. Some of the strategies covered include:

- Feature selection as a means of avoiding the *curse of dimensionality*.
- Winnowing the data set to minimize over-fitting.
- Techniques for the replacement of missing data.
- The mining of interesting rules .

Subsequent chapters apply the principles discussed here to design and assess the performance of a new decision rule induction algorithm.

IV. Overview of the GRaCCE Algorithm

The purpose of this chapter is to provide a detailed development and description of the Genetic Rule and Classifier Construction Environment (GRaCCE). This is accomplished by explaining the mechanics underlying each phase of the algorithm. A secondary goal is to furnish insight into why key design decisions were made. Throughout this chapter, diagrams and examples are liberally utilized to illustrate how the system performs decision rule induction. These are supplemented by material in Appendices B and D which, respectively, contain summaries of the algorithm parameters and notation discussed in this chapter.

4.1 Introduction to GRaCCE

The review of existing decision rule induction (DRI) algorithms (Chapter II) and pattern recognition issues (Chapter III) has provided crucial insight into the state-of-practice in these areas. The number and diversity of methods make it clear that no single method is universally recognized as superior to the others; the No Free Lunch theorem (152) has provided a theoretical confirmation of this observation. With this in mind, it makes sense that any new paradigm should (where applicable) adopt approaches found in the better techniques while avoiding their pitfalls. Based on the previously discussed material, the following are desirable DRI algorithm characteristics:

- The use of random or evolutionary search techniques.
- Classification rules are based on the natural class boundaries.
- Make a minimum number of assumptions regarding the structure of the induced rules and/or the data.
- Remove noisy or easily misclassified instances from the training data set.
- Incorporate a method of simplifying the induced rules.

- Run-time execution performance scales linearly as the size of the processed data set is increased. Additionally, the algorithm should decompose the problem such that it can be solved with a high degree of concurrency.

While some of the methods covered in Chapter II have one or more of these characteristics, none possess all of them. It would therefore be interesting (from a research standpoint) to design an algorithm that combines these attributes in a synergistic manner and measure its resulting performance. In order to be competitive with other rule induction methods, the algorithm also needs to meet the following additional design goals:

- Be a general purpose rule induction method, capable of processing a variety of different types of data sets without any prior knowledge of the problem. In particular, the system must be able to successfully process data sets that vary with respect to number of classes, features, modes and instances¹.
- Generate decision rule sets that are compact and simple with respect to those produced by other rule induction methods.
- The algorithm must achieve a level of accuracy that is roughly equivalent to that of other classification techniques when given the same data.
- The system must be robust with respect to user selectable parameters. For example, making a small change to a given parameter will not cause the results to be radically different.
- The algorithm's architecture can be easily parallelized.

GRaCCE was designed in accordance with these goals. The system harnesses the power of evolutionary search to mine classification rules from data. These rules are based on class boundary estimates generated from a simplified version of the data set. A search is then performed for combinations of these partitions that enclose class

¹The fundamental assumption here is that data of the same class is clustered together in some fashion.

Table 8 Synthetic Data Set Descriptions (2 Feature).

Name	Classes	Instances	Special Case
SYN01	3	400	Overlapping classes.
SYN02	2	200	Iterative partition generation.
SYN03	5	1000	Solvable with global partitions.
SYN04	2	1000	Multi-modal data

homogeneous (CH) regions in the data. These regions are refined further and used to generate a final rule set for classifying the data. Thus, the GRaCCE algorithm is defined in terms of six distinct phases: preprocessing, partition generation, data set approximation, region identification, region refinement and partition simplification. The sections that follow provide an in-depth description of GRaCCE in terms of these phases. In addition, examples are provided which illustrate how GRaCCE processes special conditions (modeled by the synthetic data sets described in Table 8).

4.2 Preprocessing Phase

The purpose of this first phase is to prepare the data for subsequent phases of the algorithm. The two primary preprocessing operations that accomplish this (feature selection and winnowing) are described in the subsections that follow. Of these two operations, only winnowing is mandatory. Note that other preprocessing operations (such as generating missing data) are not part of GRaCCE due to the unique requirements of each data set.

4.2.1 Feature Selection. In this first phase of the algorithm, we perform feature selection to reduce the data set's dimensionality. Recall from discussion in Section 3.2 that removing unnecessary features is one of the surest ways to mitigate the *curse of dimensionality* (8) during training and also simplifies the structure of the decision rules. GRaCCE offers a number of feature selection algorithms including a deterministic forward-search (DFS), a GA-based search, or a hybrid of the two. The DFS method sequentially searches for the smallest feature set that minimizes

the error rate of a k Nearest Neighbor (kNN) algorithm. Starting with an empty feature set, DFS chooses the feature that most reduces the kNN error. The selected feature is added to the feature set. On subsequent iterations, candidate features are evaluated in conjunction with the existing feature set. This process continues until the error rate cannot be improved by adding additional features to the set. At this point, the DFS returns the current feature set.

While effective, the DFS is not without shortcomings. For example, its time complexity is quadratic (i.e., $O(nd^2)$) with respect to the number of features (d). In addition, its greedy approach may cause superior combinations of features to be overlooked. To compensate for these factors, DFS is augmented by a relatively simple GA-based approach developed by Sklansky and Siedlecki (124). In this technique, the GA's binary chromosome structure is organized by mapping each feature to a separate gene. The value of each gene's allele determines if the associated feature is ignored (0) or utilized (1). The GA's objective function evaluates the feature set represented by each individual; the fittest individuals are those that achieve the best classification accuracy on the kNN algorithm with the fewest enabled features. When the GA terminates, the dimensionality of the data is reduced by eliminating those features disabled in the fittest individual.

The hybrid method combines elements of the two previous techniques. It begins by performing an abbreviated DFS to identify the best d_s features (where $d_s \ll d$). In preparation for a more comprehensive feature search, a GA population is randomly generated. The bits representing the DFS features are then set in each member of the GA population. The motivation here is to better focus the GA search using a promising schema. Once the population is modified, the GA is executed as indicated in the previous paragraph.

4.2.2 Winnowing. Class overlap within a given data set can impede the rule induction process and degrade the quality of the generated rule set. It is there-

fore desirable to make the classes piecewise, linearly separable. Wagner’s *edited nearest neighbor* technique (147) provides an elegant way of accomplishing this goal. Wagner’s method removes data points from the training set that are misclassified by a kNN algorithm. This process continues until it converges to a subset of the data with no classification error. Devijver and Kittler (27) proved that this procedure is asymptotically Bayes-optimal with respect to the original data set. Figure 22 illustrates how this approach can improve class separation for a synthetic data set. Winnowing can be performed on either the full feature set or on a specified feature subset. As an added benefit, there is evidence showing that this procedure contributes to a reduction in rule set size (refer to Section 3.6).

4.3 Partition Generation Phase

The objective of this phase is to generate a set of partitions sufficient to separate the classes in a given data set. Let Ω denote the data set and ω_i be the subset of Ω belonging to the *i*th class. A class partition is any hyper-plane separating two distinct classes (ω_i and ω_j). We now extend the notation, such that Γ represents the set of all generated partitions and γ_i denotes the subset of partitions associated with the *i*th class. Because these partitions are the basis for evolving CH regions, it is extremely important that they reflect the natural class boundaries. To ensure this, GRaCCE uses a two pronged approach such that both global and local class partitions are generated. The rationale for and specifics of this approach are discussed in the sections that follow.

4.3.1 Fundamentals of Partition Estimation. In order to generate a partition separating two classes, it is necessary to estimate the probability distribution of each class from an appropriate set of samples. The sample set can vary in size from a few points to all points in each class. The statistics required for the class distribution estimate are the mean (μ) and covariance matrix (Σ). From these, partition separating the classes can be computed. We define such a partition as a

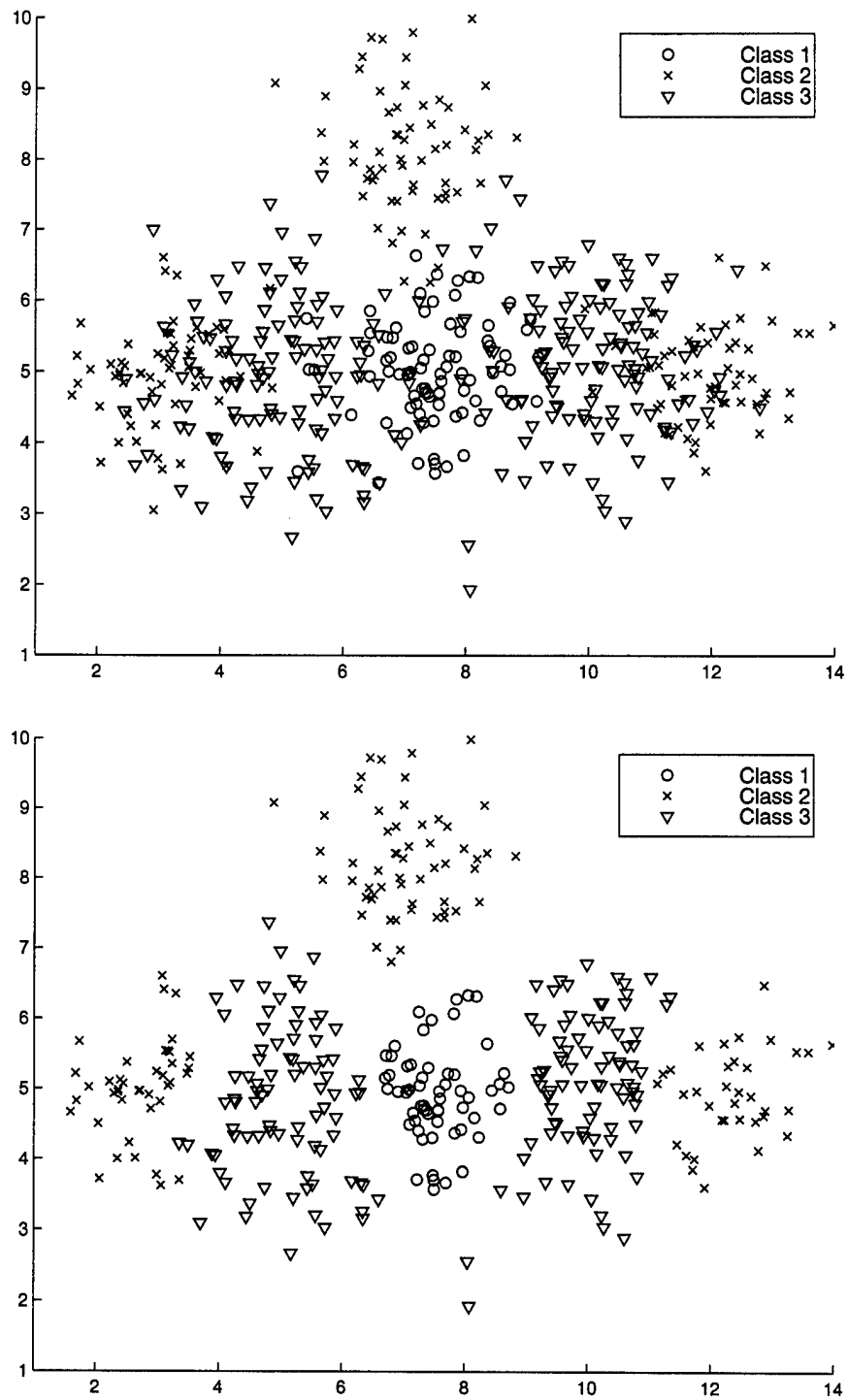
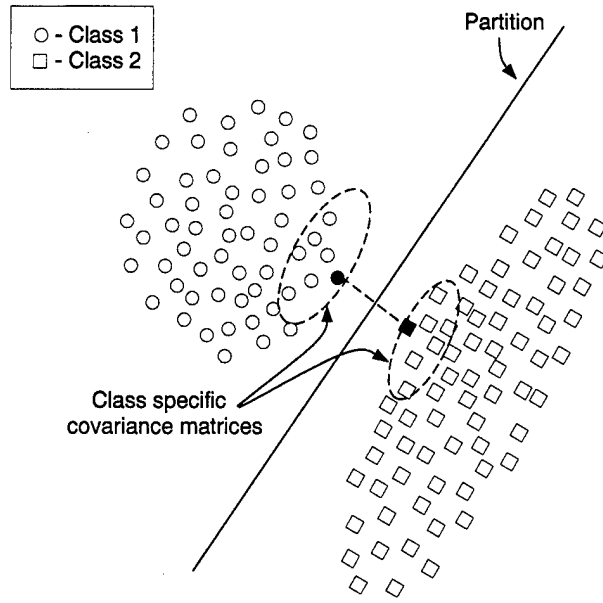


Figure 22 Data Set SYN01 before (above) and after (below) the Winnowing operation.



Neighborhood around each boundary point in the identified boundary point pair is used to compute the covariance matrices needed to estimate the Bayes optimal decision boundary.

Figure 23 Partitions are generated to separate boundary point pairs.

hyper-plane function (h) composed of an anchor point (X_0) through which h passes and a vector normal to the plane \vec{v}_N ; these components are defined by Equations 13 and 14, respectively. If μ_i and μ_j are on different sides of the decision boundary, then X_0 lies along the line connecting these points such that $h(X_0) = 0$. Thus, Equation 15 can be used to determine the position of a given point X with respect to h . Figure 23 shows a partition generated to separate two classes. While other non-deterministic methods, such as perceptrons (10), could be employed to generate the partitions, this deterministic method consistently yields a decision boundary that better approximates the Bayes decision boundary.

$$X_0 = uV + V_0 \quad (13)$$

where:

$$\begin{aligned}
V &= \mu_i - \mu_j. \\
V_0 &= \mu_i \\
u &= \frac{t-c}{b} \text{ if } a = 0, \\
u &= \frac{-b \pm \sqrt{b^2 - 4a(c-t)}}{2a} \text{ and } 0 \leq u \leq 1 \text{ if } a \neq 0, \\
a &= \frac{1}{2} V'(\Sigma_i^{-1} - \Sigma_j^{-1})V, \\
b &= V_0'(\Sigma_i^{-1} - \Sigma_j^{-1})V, -(\mu_i' \Sigma_i^{-1} - \mu_j' \Sigma_j^{-1})V, \\
c &= \frac{1}{2} V_0'(\Sigma_i^{-1} - \Sigma_j^{-1})V_0 - (\mu_i' \Sigma_i^{-1} - \mu_j' \Sigma_j^{-1})V_0 + \frac{1}{2}(\mu_i' \Sigma_i^{-1} \mu_i - \mu_j' \Sigma_j^{-1} \mu_j) + \frac{1}{2} \ln \frac{|\Sigma_i|}{|\Sigma_j|}, \\
t &= 0, \text{ assumes } P(i) = P(j).
\end{aligned}$$

$$\vec{v}_N = (\Sigma_i^{-1} - \Sigma_j^{-1})X_0 + (\Sigma_i^{-1}\mu_i - \Sigma_j^{-1}\mu_j) \quad (14)$$

$$h(X) = \vec{v}_N(X - X_0)' \quad (15)$$

4.3.2 Global Partition Generation. Global partitions are those generated from sample sets composed of all data for a given class. In this mode, GRaCCE computes a single, linear partition separating each pair of classes. These types of partitions are useful because they reflect the shape of the entire distribution of each class (estimated from its covariance matrix) (29); this is something locally derived partitions cannot do. Using this approach on a data set with m classes results in a set of $m(m-1)/2$ partitions. In some cases, the partitions generated in this mode can sometimes be sufficient to completely isolate each class. Figure 24, which contains only global partitions, is an example of such a case. Observe that it is still possible

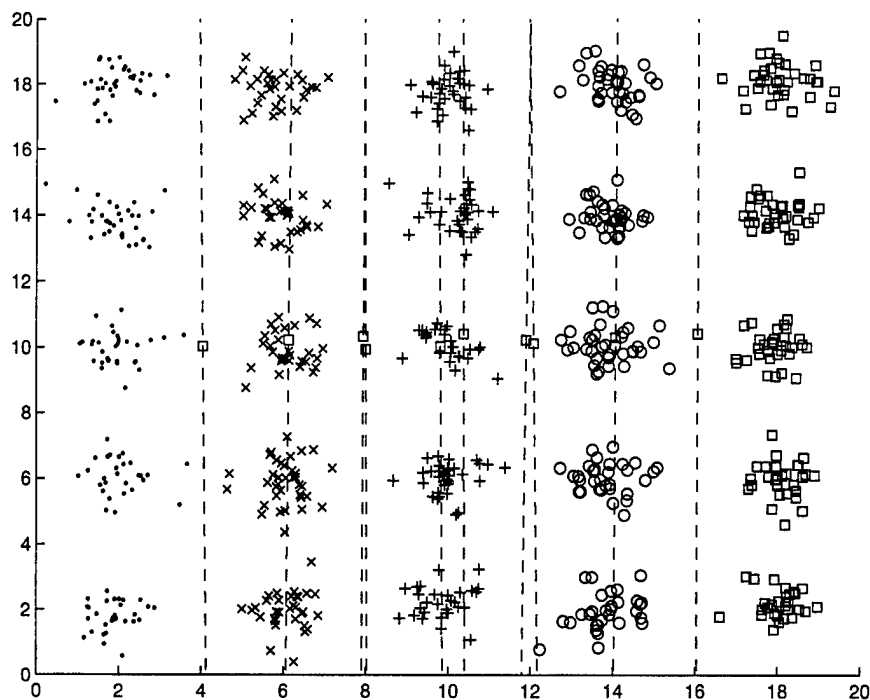


Figure 24 Global Partition Generation for Data Set SYN03.

to seal each class off from the rest of the data using the sparse set of generated partitions.

4.3.3 Local Partition Generation. As might be expected, only the most trivial of classification problems can be solved using global partitions alone. In fact, for some data sets they may be totally unusable. Consider the data set depicted in Figure 25. Since the modes of both classes are interleaved in a checkerboard pattern, their mean and covariance matrices are virtually identical. Consequently, any global partitions generated from this set is meaningless. It therefore becomes necessary to generate other types of partitions to augment the global set.

The solution to this problem is to generate additional partitions which approximate class boundaries between adjacent boundary points of differing class. Partitions in this category are applicable only to a local region of the data set. When there is significant class overlap, as in the original SYN01 data set (see Figure 22), many

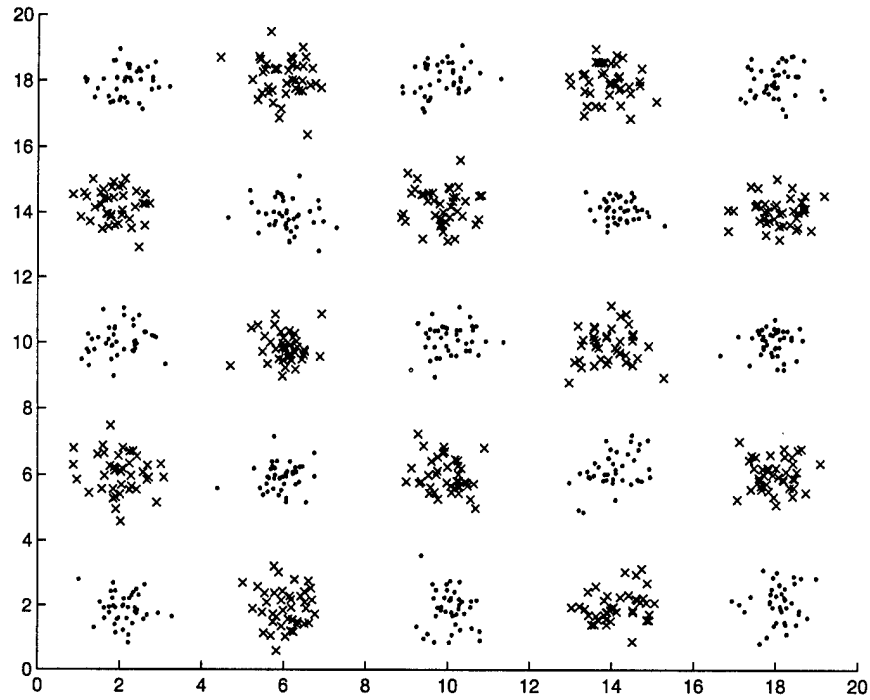


Figure 25 The interleaving of class modes in SYN04 renders global partitions useless.

partitions are generated which do not reflect the natural class boundaries. Fortunately, the winnowing operation reduces the number of candidate boundary points by making the classes linearly separable. As a result of this operation, the number of generated partitions is much lower than it would otherwise have been. In addition, the quality of the partition estimates is substantially improved because they are based on data with a solid degree of membership in each class.

Definition 1 - Boundary Point. Let the function D be the Euclidean distance between two arbitrary feature vectors. Let z and y be members of classes ω_i and ω_j , respectively. Instance y is a boundary point with respect to z iff $D(z, y) = \min\{D(z, \zeta) : \zeta \in \omega_j\}$. Let Λ and λ_t denote the set of boundary points for the entire data set (Ω) and some class (ω_t), respectively.

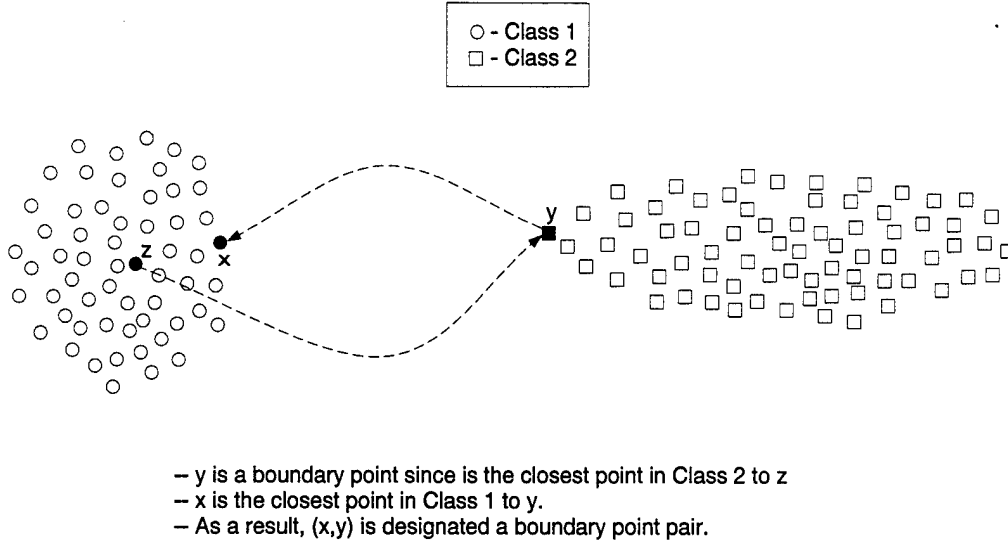


Figure 26 Generation process for boundary point pairs.

Definition 2 - Boundary Point Pair. Let x be a points in ω_i and y be a boundary point in ω_j such that $D(x, y) = \min\{D(\zeta, y) : \zeta \in \omega_i\}$. When this condition is satisfied, points x and y are said to be a boundary point pair.

The first step in generating local partitions is locating the boundary points (as described in Definition 1) for each class in the winnowed data set. Next, a list of boundary point pairs is compiled using Definition 2. Each pair consists of two points, (x, y) belonging to different classes ($x \in \omega_i$ and $y \in \omega_j$). Figure 26 provides a visual illustration of these definitions. This approach enables GRaCCE to generate class partitions for highly multi-modal data configuration; examples that support this contention are shown for data sets SYN01 and SYN04 in Figures 27 and 28, respectively.

4.3.4 Sufficiency of the Partition Set. In order for the GRaCCE algorithm to succeed, the generated partition set must be sufficient to isolate each boundary point in a CH region (of the same class). The upper half of Figure 30 illustrates a case where the initial partition set does not meet this requirement. Note that the position of the middle two clusters is such that Definitions 1 and 2 fail to yield

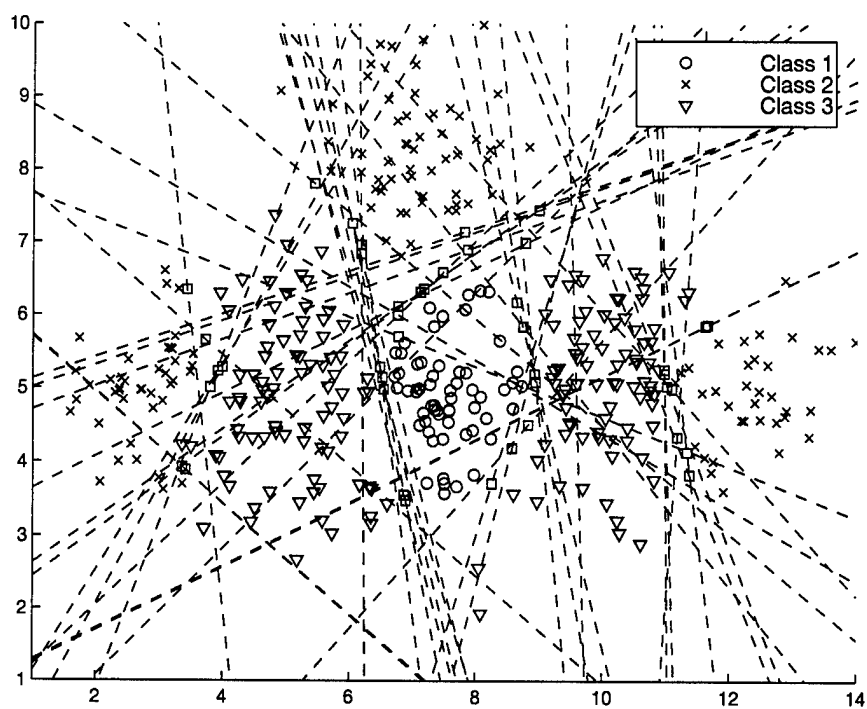


Figure 27 Set of Generated Partitions for SYN01.

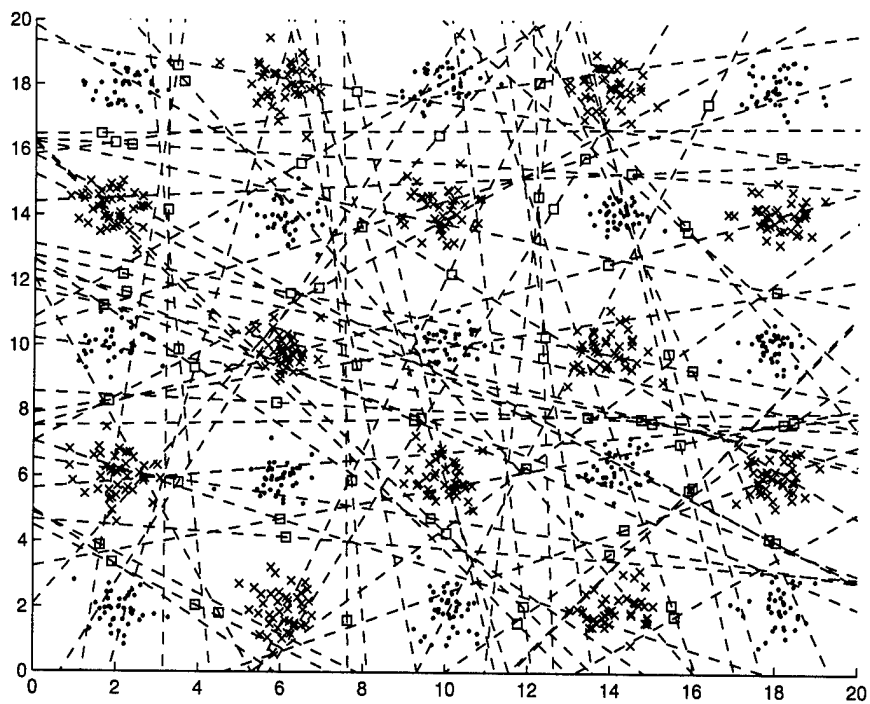


Figure 28 Set of Generated Partitions for SYN04.

```

- Given the sets:  $\Omega$ ,  $\Gamma$ , and  $\Lambda$ .
 $b = \text{Next}(\Lambda)$ ;
while ( $b \neq \emptyset$ ) do
  - Create a new partition set,  $\Gamma'$ , such that:
     $\forall h \in \Gamma, h(b) \geq 0$ .
  - Compile a list of data points,  $S$ , composed of all  $x \in \Omega$  such that:
     $\forall h \in \Gamma', h(x) \geq 0$ .
  if ( $\forall x \in S, \text{class}(x) = \text{class}(b)$ ) then:
     $b = \text{Next}(\Lambda)$ ;
  else
    - Generate a set of boundary points and partitions for  $S$ .
    - Append these to  $\Lambda$  and  $\Gamma$ , respectively.
  end; %while
return  $\Gamma, \Lambda$ .

```

Figure 29 Iterative Partition Generation Algorithm.

a partition separating them; consequently, these clusters cannot be enclosed. To remedy this situation, the partition generation process must continue until sufficient partitions are generated to enclose each boundary point in its own CH region.

The process to accomplish this (described by the pseudocode in Figure 29) is performed as follows: given a boundary point $b \in \lambda_t$, all partitions associated with the target class (γ_t) are used to form a convex region isolating b . If the subset of data in this region contains instances not belonging to ω_t , additional boundary points and partitions are generated to rectify this condition². This process iterates until a CH region can be formed for each boundary point using the existing partition set. The lower half of Figure 30 shows the additional partition added as a result of this procedure. As a result, the final partition set is sufficient (but not uniquely so) to separate all boundary points of differing class.

²These additional boundary point and partitions are generated using *all* the entire winnowed data set - not just the enclosed subset.

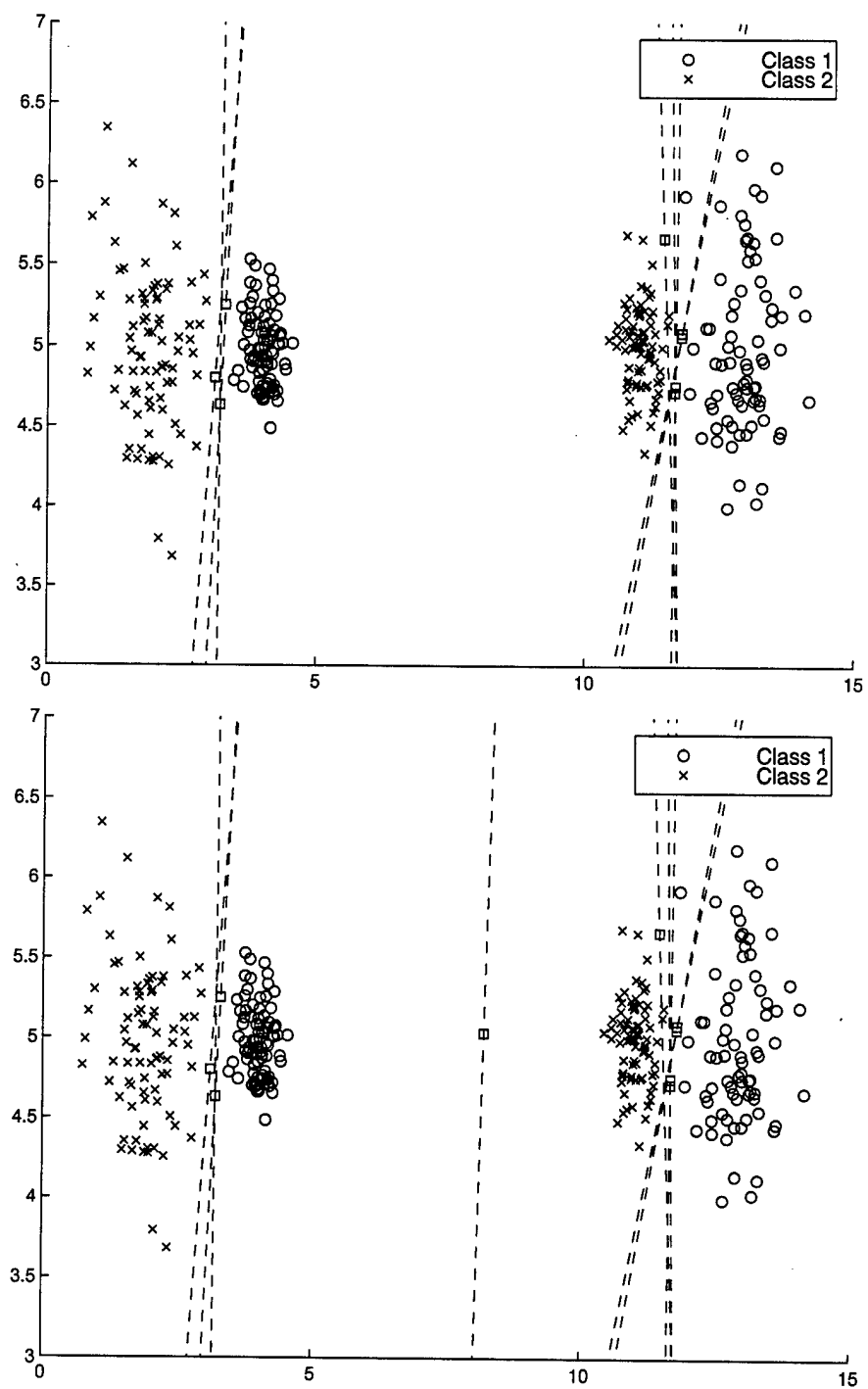


Figure 30 Initial (above) and Final (below) Partition Sets for Data Set SYN02.

Definition 3 - Boundary Point Weighting Function (ρ). Given a class ω_t , and a boundary point $b \in \lambda_t$, the weight for b is computed as:

$$\rho(b) = \frac{1}{|\omega_t|} \sum_{x \in \omega_t} \chi(x), \text{ where: } \chi(x) = \begin{cases} 1, & \text{if } D(b, x) = \min\{D(\zeta, x) : \zeta \in \lambda_t\} \\ 0, & \text{otherwise} \end{cases}$$

4.4 Data Set Approximation

Many pattern recognition techniques take advantage of the fact that not every point in a data set is essential to classification (22). Since the boundary points in the winnowed data set lie along the borders of the CH regions we seek, the data set can be approximated by Λ . This is conceptually analogous to the way a skeleton outlines the shape of a body. Since the set of Λ is typically a small fraction of the total number of training points, using it as the basis for the CH region search results in a significant acceleration of the algorithm. In order to make the best use of Λ , we complement it with an estimate of the relative importance of a given boundary point (b) within its assigned class. This estimate is provided by a weighting function (ρ) which is specified in Definition 3. The value returned by $\rho(b)$ represents the relative density of class data around boundary point b .

An alternative to this method is the PLEASE system (74), which is a Pittsburgh-style (127) genetic-based machine learning system for learning class *prototypes*. A class prototype is a set of exemplars which can be used as a substitute for the class data in a kNN algorithm. In particular, the GA uses a variable length, real-valued chromosome to evolve a set of prototypes for a given data set. As might be expected, the computational complexity of this approach becomes prohibitive as the dimensionality of the data set increases. Indeed, the authors hint at this problem and the PLEASE system is only demonstrated on relatively simple, two-dimensional data sets.

Definition 4 - GA Chromosome Structure (I). This is the template for the binary chromosome (\vec{a}) used in the GA search for a CH region of class t ($\vec{a} \in I$). The characteristics of I are determined by boundary point (b) chosen as the focal point for the search. It consist of three components: l, f_m, f_o . The l component is the length of the binary vector, which equals the number of partitions used in the search. The f_m component is the mapping function between chromosome \vec{a} and the partitions in γ_t it represents. The f_o component is the function that returns the orientation of each partition (value = ± 1) with respect to b . Thus the j th partition (h_j) in the chromosome is equivalent to $f_o(j)f_m(j)$, such that $h_j(b) \geq 0$.

Definition 5 - Region. Given a binary chromosome $\vec{a} \in I$, then the region represented by \vec{a} ($[\vec{a}]$) is defined as: $[\vec{a}] = \{p \in \mathbb{R}^d | h_j(p) \geq 0, \forall j \ni \vec{a}(j) = 1\}$.

4.5 Region Identification Phase

The region identification (RI) phase is the centerpiece of the GRaCCE algorithm. It consists of a series of searches to identify the set CH regions (R) within a given data set. The primary resources required for this search are the generated boundary point and partition sets. The objective of each search is to find a set of partitions that isolate boundary points belonging to a given class with a user-specified degree of purity (δ_{min}). Each search begins by selecting a target class ω_t . To find a CH region in class ω_t , the boundary point (b) with the greatest weight is chosen from λ_t as the focal point for the search. The pseudocode describing this phase can be found in Figure 32.

Genetic Algorithms are used as the primary search engine for the RI phase. There are a number of compelling reasons for this choice. Perhaps the most fundamental is that GAs have proven to be extremely effective in finding good solutions for combinatorial optimization problems (41). For most optimization problems, the search space is large enough to make exhaustive search methods impractical (95). Unlike search algorithms which incrementally develop an optimal solution, such as

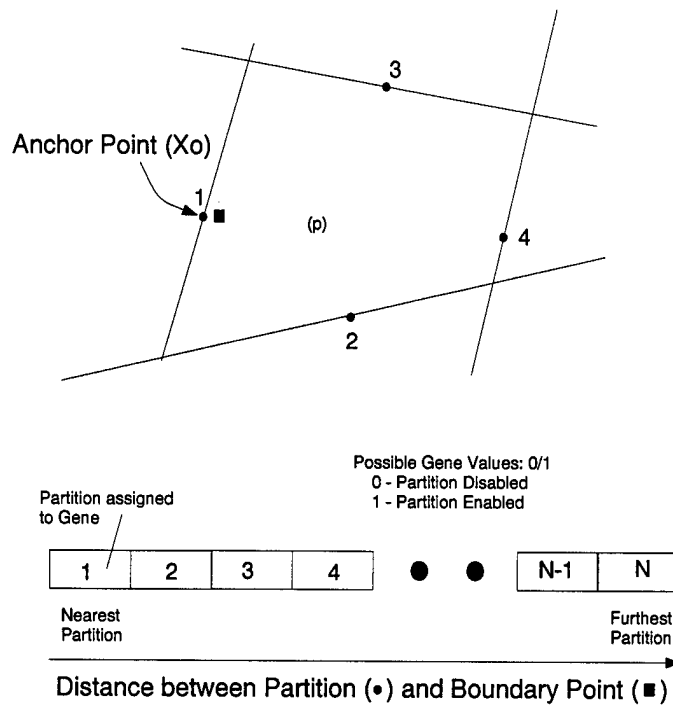
A^* (109), GAs maintain a ready pool of fit solutions. As a result, GAs can return the best, complete solution at any given time. This characteristic is important because GRaCCE may not have a great deal of time to spend on each CH region search.

In preparation for each search, a GA chromosome structure (I) is organized as described in Definition 4 for each chosen boundary point. The purpose here is to construct a template that enables each individual in the GA population to represent a CH region. This task is accomplished by assigning a specific partition to each gene in the chromosome. The binary allele of the gene denotes the partition's status (0 if disabled, 1 if enabled). Thus the region defined by each individual ($\vec{a} \in I$) in the GA population (P) is determined by which of its partitions are enabled. Figure 31 provides an illustration of how the class partitions are represented in the GA chromosome structure. In addition, Definition 5 is used to determine if a given point is a member of the region, as represented by some individual \vec{a} .

For any given CH region search, there are a number of steps that can help limit the size of the search space. First, the GA chromosome need only include partitions in γ_t ; all others are irrelevant. Partitions in this set are sorted in ascending order based on the distance of their anchor point to b . This arrangement enables the search to be restricted to the closest partitions if the number under consideration exceeds a specified threshold. The implicit assumption here is that the further a partition is from b , the less likely it is that the partition defines a CH region containing b . Lastly, all partitions are assigned an orientation transform (of ± 1) to insure that $h(b) \geq 0$; this transform is incorporated into the mapping function, f_m . Since this makes it unnecessary to represent a partition's correct orientation in the GA chromosome structure, the search space is significantly reduced.

Once I is defined, a deterministic search is performed to find a CH region (\vec{z}) that, at a minimum, contains b ; the pseudocode for this algorithm is shown in Figure 33. Because this procedure is greedy, there is no assurance that \vec{z} is optimal in terms of fitness. It does, however, serve as a good starting point for each

Genetic Algorithm Chromosome Layout



Given a chromosome structure, use Definition 4 to determine if a point (p) is within the region represented by a given individual.

Figure 31 GA Chromosome Structure.

evolutionary search. Using a variation of Eshelman's random restart technique (31), GA population is initialized such that half of its individuals are mutations of \vec{z} ; the remaining individuals are randomly generated. Early experiments showed this approach produced better solutions in a shorter amount of time (as compared to using a completely random initial population). Although this practice biases the GA toward a specific area of the search space, as a practical matter GRaCCE cannot afford to conduct an extensive search for every region. Approaches such as Fast Messy GAs (43) are extremely effective, but have too much overhead for this problem. Thus, the GA performs a limited search for a better solution (\vec{x}) than the original. The effectiveness of using a stochastic search to improve an existing solution was previously demonstrated by the OC1 decision tree algorithm (94). If the GA cannot accomplish this within a fixed number of generations (q), it defaults back to \vec{z} .

The GA searches for solutions that minimize the objective function (Φ) defined by Equations 16 thru 18. Within this function, $\phi_1(\vec{a})$ is the proportion of class ω_t data contained in the region defined by \vec{a} . The term $\phi_2(\vec{a})$ denotes the complexity of the region in terms of the number of partitions that bound it. As indicated earlier, the boundary point weights are used to estimate the proportion of data in \vec{a} belonging to each class; we denote this result for the i th class as $\delta_i(\vec{a})$. If $\delta_i(\vec{a}) < \delta_{\min}$, we compute fitness using a graded penalty function. This strategy drives the GA to seek a solution that maximizes coverage of the target class with a minimum of complexity. The graded penalty function enforces the user's homogeneity requirement without losing good schema contained in unacceptable solutions (126). Figures 34 through 36 show the SYN01 class partitions chosen by GRaCCE to isolate classes 1 through 3, respectively.

$$\phi_1(\vec{a}_k) = 1 - \sum_{i=1}^{|\lambda_t|} \left\{ \begin{array}{ll} \rho(b_i), & \text{if } b_i \in [\vec{a}_k] \\ 0, & \text{otherwise} \end{array} \right\} \quad (16)$$

$$\phi_2(\vec{a}_k) = \min(0.05, 1/l) \sum_{j=1}^l \vec{a}_k(j) \quad (17)$$

$$\Phi(\vec{a}_k) = \left\{ \begin{array}{ll} \phi_1(\vec{a}_k) + \phi_2(\vec{a}_k), & \text{if } \delta_{\min} \leq \delta_t(\vec{a}_k) \\ 2 - \delta_t(\vec{a}_k) + \phi_2(\vec{a}_k), & \text{otherwise} \end{array} \right\} \quad (18)$$

When the search has completed, the fitness of the best region found (\vec{x}) is evaluated. If its fitness falls below 1.0, then the training data enclosed within \vec{x} are formally assigned to it and the region is then appended to R ; otherwise, b and \vec{x} are discarded. After each search, the list of unassigned boundary points contained in λ_t is updated. Additional CH region searches are conducted until all boundary points in λ_t have been assigned (i.e., $\text{unassigned}(\lambda_t) = \emptyset$). The algorithm then repeats this procedure for all remaining classes. Once all classes are exhausted, the final set of regions is returned.

4.6 Region Refinement Phase

The purpose of this phase is to improve the regions found in the previous phase. To accomplish this, each region (\vec{a}) is evaluated in three ways. First, the CH regions are refined by removing extraneous partitions. Each boundary enabled in \vec{a} is tested to ensure it contributes to $\Phi(\vec{a})$. If removal of the boundary causes no degradation in fitness, it is no longer mapped to the region. When all regions have been processed, all unused partitions are expunged from the system.

The next step is to filter out regions which are both small (in terms of coverage) and defined by a disproportionate number of partitions. To this end, a region utility ratio (RUR) is computed for each region using Equation 19. In general, this metric yields small values (less than 0.01) for regions of poor quality, with values increas-

```

% SYMBOLS:
b  : Selected boundary point.
l  : Number of generated partitions.
m  : Number of data set classes.
p  : Size of GA population.
q  : Convergence window size (in generations).
P  : GA population.
R  : Region set.
 $\lambda_t$  : Set of unassigned boundary points for  $\omega_t$ .
I  : GA Chromosome structure.

% PSEUDO-CODE:
R =  $\emptyset$ ;
for t = 1 to m do,
    while (unassigned( $\lambda_t$ )  $\neq \emptyset$ ) do
        b = max(weights(unassigned( $\lambda_t$ )));
        - Organize GA Template, I.
        [z] = Greedy_Search(I);
        - Initialize P = { $\vec{a}_1 \dots \vec{a}_p$ }  $\in I$ .
        - Evolve P for q generations.
         $\vec{x}$  =  $\vec{a}_k \ni \Phi(\vec{a}_k) = \min(\Phi(P))$ 
        if ( $\Phi(\vec{x}) \geq \Phi(\vec{z})$ ) then
             $\vec{x}$  =  $\vec{z}$ ;
        else
            - Search until  $\Phi(\vec{x})$  is constant for  $q^+$  generations.
            end; %if
        - Update R and  $\lambda_t$  based on  $\vec{x}$ .
        end; %while
    end; %for t
return R;

```

Figure 32 Region Identification Phase Algorithm.

```

function  $\vec{z}$  = Greedy_Search( $I$ )
 $\vec{z} = \{1\}^l$ ;                                % Enable all partitions.
 $s = \Phi(\vec{z})$ ;                                % Compute the initial fitness.
% Disable any partitions which do not improve fitness.
for  $i = 1$  to  $l$  do
     $\vec{z}(i) = 0$ ;
    if ( $\Phi(\vec{z}) > s$ ) then
         $\vec{z}(i) = 1$ ;
    else
         $s = \Phi(\vec{z})$ ;
    end; %if
end; %for  $i$ 
return  $\vec{z}$ ;

```

Figure 33 Greedy Search Algorithm for the Initial CH Region.

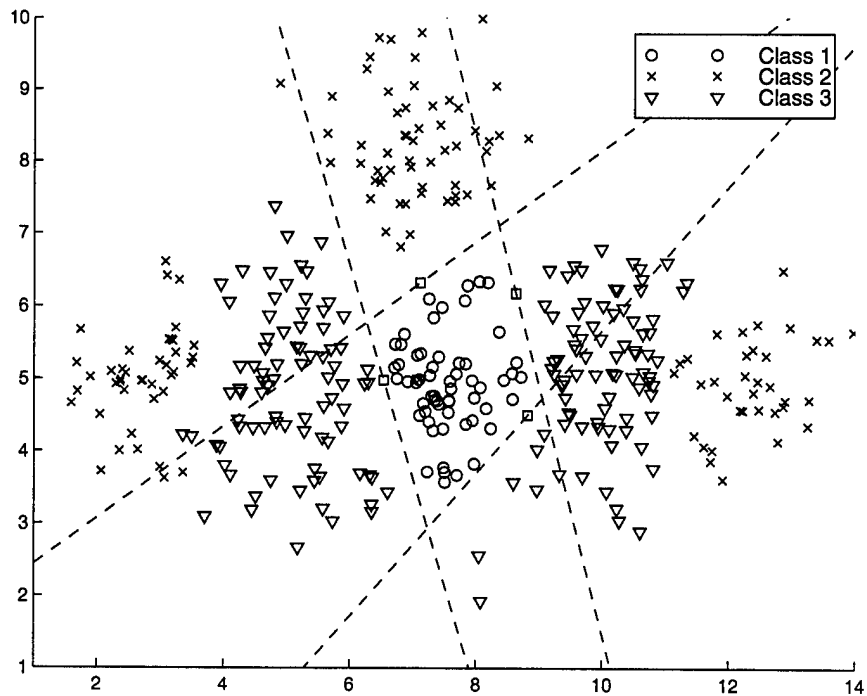


Figure 34 SYN01 Partitions selected for Class 1.

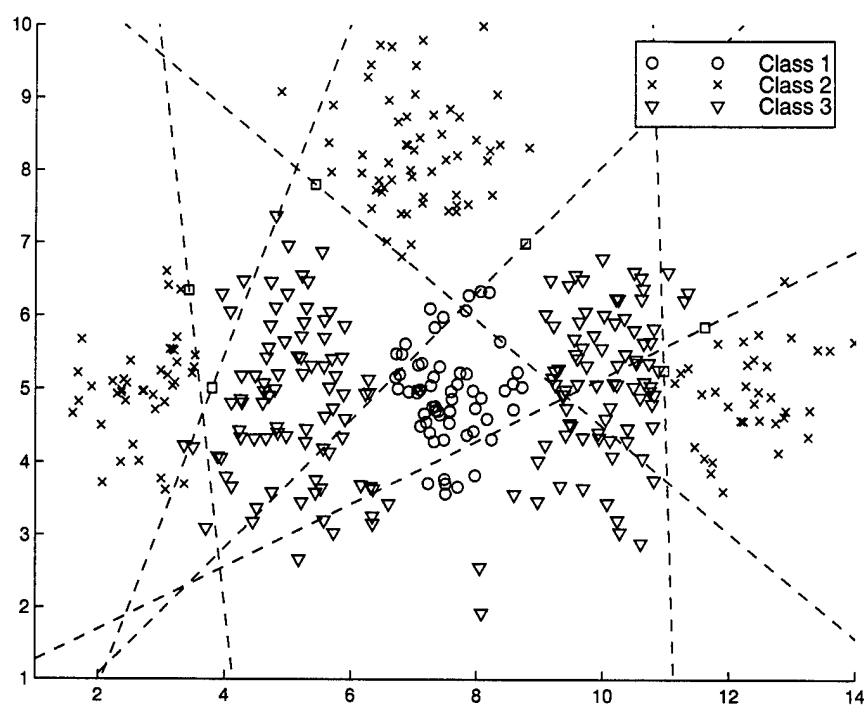


Figure 35 SYN01 Partitions selected for Class 2.

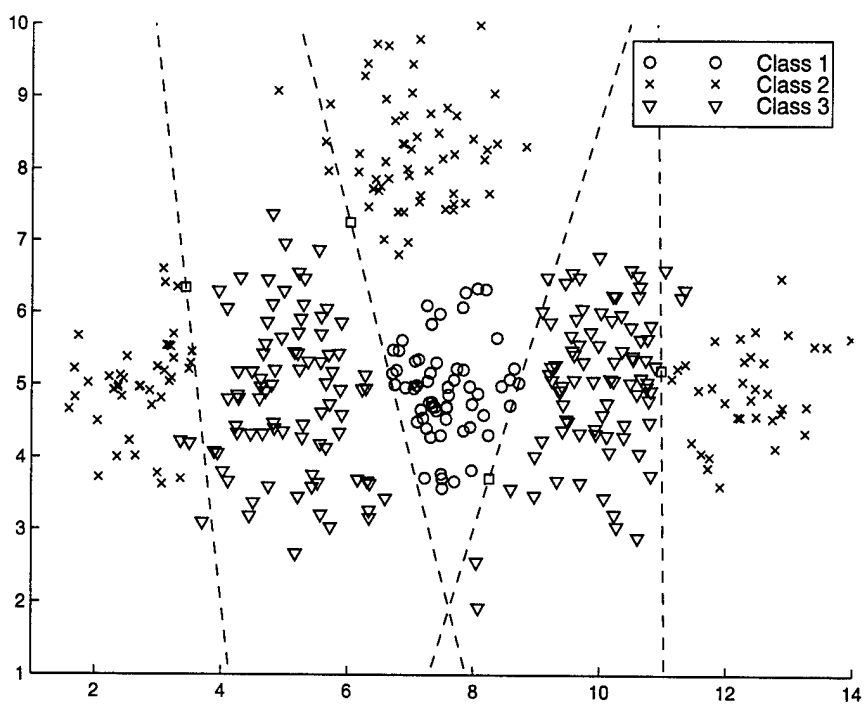


Figure 36 SYN01 Partitions selected for Class 3.

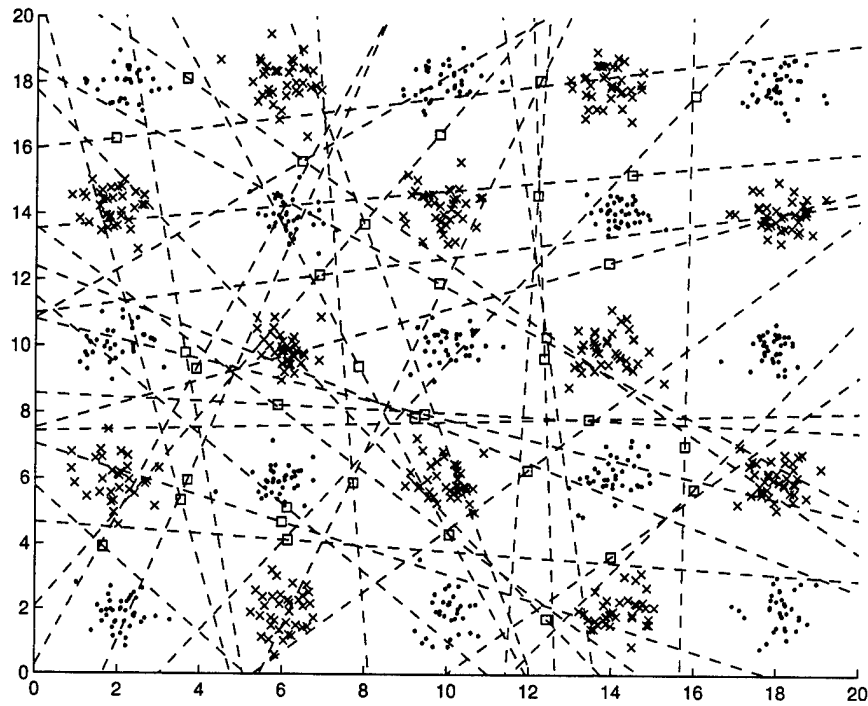


Figure 37 Partitions selected for SYN04.

ing with region quality. The RUR for each region is compared to a user-specified threshold (RUR_{min}); regions whose RUR fall below this threshold are eliminated. Removing these regions is akin to the procedure used to prune decision trees in that it helps avoid over-training while reducing unnecessary complexity. Unlike pruning, however, it takes into account the complexity of the region. Another difference is that instances in deleted regions become *orphan* data in that they are not assigned to any class. While this may appear to unnecessarily increase the error rate, a method for classifying orphan data is discussed in Section 4.9.2. Of course, a simpler alternative is to retain all found regions and let the user filter out those deemed inconsequential (due to insufficient membership).

After completion of the first two procedures, the set of regions is considered to be complete. The final step in the refinement process is to recompute the covariance matrix of each region based on the data points assigned to it. With the updated covariance matrix, Equation 15 is used to adjust the orientation of each partition

to better reflect the boundaries of the regions they separate; only those adjustments that improve the classification accuracy of the training set are retained.

At the conclusion of this phase, we are left with a set of regions, a list of partitions, a mapping of regions to partitions and a mapping of data to regions. Because each region is separated from the rest of the data space by the partitions that bound it, each region can be treated as a rule for classifying the data and each of its partitions as conditions of that rule.

$$\text{RUR}(\vec{r}_i) = \frac{\delta_t(\vec{r}_i) * \kappa_t(\vec{r}_i)}{\frac{1}{|\Gamma|} \sum_j |\vec{r}_i(j)|} \quad (19)$$

where:

- t : Target class index for the i th region.
- κ_t : Proportion of class ω_t enclosed by the i th region.

4.7 Partition Simplification Algorithm

In the baseline GRaCCE algorithm, all partitions are specified with respect to all d dimensions. For large feature sets, this can result in rules with very complex conditions. It is therefore desirable to simplify these partitions to the maximum degree possible without significantly degrading the accuracy of the rule set. This goal is accomplished using a partition simplification algorithm (PSA) based on the *backward deletion* process developed by Breiman (11) for the CART algorithm.

The PSA (summarized in Figure 38) eliminates those terms that are not essential to the partition. This determination is based on the degree to which classification error (ϵ) is increased by removing a given term from a partition. The increase in ϵ caused by removing the j th term from the partition is denoted as $(\epsilon)_j^+$; this difference is measured for each non-zero term in the partition. Two user specified parameters

provide the thresholds needed to determine if the partition can be simplified further based on these measurements. The first parameter (α) limits the absolute increase in error resulting from a given change. The second parameter (β) provides a ratio of $(\epsilon)_{min}^+$ to $(\epsilon)_{max}^+$ for a given partition. The actual ratio must be less than this threshold in order for the *min* term to be removed. When the PSA completes, the proportion of partition terms removed from the rule set is measured; this metric is referred to as the degree of partition simplification (DPS).

This modified algorithm differs from Breiman's in several important ways. The first is the scope of testing required for each changed partition. Since CART is a decision tree algorithm, each partition implements a different node in the tree. This structure enables Breiman to restrict testing to a subset of the data assigned to a given node. Unfortunately, this approach is not feasible here since a given partition may bound multiple regions. As an alternative to computing ϵ based on the entire training set, the set of weighted boundary points is used to approximate the overall error. This technique is already used to evolve the CH regions and is far more efficient than using the entire data set.

Second, Breiman attempts to optimize the partition's anchor point (X_0) in response to every change. Our early experiments indicated that this is an expensive operation which has minimal impact on the end result. Consequently, only the vector portion (\vec{v}) of each partition (see Equation 14) is modified in this approach. Lastly, while Breiman uses only the β parameter in his process, an additional threshold parameter (α) is used here to limit the absolute degradation in accuracy resulting from a given change. This added feature gives the user more control over the simplification process.

The principle drawback of this method, however, is its extremely large time complexity (refer to Table 11). As a result, using this technique for high dimensionality data sets is prohibitive. Another possible approach is to sort the terms in a given partition's unit vector in order of increasing size. Starting with the smallest

% SYMBOLS:

ϵ : Baseline error rate.
 \vec{v}_i : Vector normal to i th partition in set Γ
 $(\epsilon)_{max}^+$: Maximum increase in error relative to ϵ .
 $(\epsilon)_{min}^+$: Minimum increase in error relative to ϵ .
 j_{min} : Term index corresponding to $(\epsilon)_{min}^+$.
 α : Minimum allowable increase in error.
 β : Minimum ratio of $(\epsilon)_{min}^+$ to $(\epsilon)_{max}^+$.

% PSEUDO-CODE:

```
- Compute  $\epsilon$ .
 $i = 1$ ;
while  $i \leq |\Gamma|$  do
  - Initialize  $(\epsilon)_{min}^+$  and  $(\epsilon)_{max}^+$  for  $\vec{v}_i$ ;
  for  $j = 1$  to  $d$  do
    if  $(\vec{v}_i(j) \neq 0)$  then
       $\vec{v}_i(j) = 0$ ;
      - Compute  $(\epsilon)_j^+$  (for  $j$ th term).
      - Update  $(\epsilon)_{min}^+$  and  $(\epsilon)_{max}^+$  based on  $(\epsilon)_j^+$ .
      - Restore each term in  $\vec{v}_i$  to its baseline value.
    end; %if;
  end; %for;
  if  $((\epsilon)_{min}^+ < \alpha)$  and  $((\epsilon)_{min}^+ \leq \beta(\epsilon)_{max}^+)$  then
    % Update the baseline variables.
    - Update  $\Gamma$  such that  $\vec{v}_i(j_{min}) = 0$ .
    - Update  $\epsilon$ .
  else
    % Process the next partition.
     $i = i + 1$ ;
  end; %if
end; %while;
return  $\Gamma$ ;
```

Figure 38 Partition Simplification Algorithm.

term, each term can be evaluated in terms of its contribution to the overall error rate. Those terms found to have minimal (under a given threshold) or negative contributions can then be eliminated. While the simplistic, single pass search is computationally inexpensive, the tradeoff is that it will probably yield lower quality partitions than the more extensive search currently implemented.

4.8 A Decision Rule Set Example - The Iris Data Set

An example of a decision rule set generated using the GRaCCE induction process is now presented. For simplicity, Fisher's Iris data (34) was chosen as the example data set due to its small size. The Iris data consists of four measurements (features) describing three different types of irises (Setosa, Versicolour and Virginica); these features are: sepal length, sepal width, petal length and petal width. It was determined through the feature selection process that the most salient features were petal length and petal width; thus, the version of Iris input to GRaCCE contained only this reduced feature set.

After processing the Iris data, GRaCCE produced a decision rule set consisting of two partitions and three CH regions (one for each Iris class). The report containing this information is similar in format to the example report in Appendix C. While not reproduced here in its entirety, the portion of the report needed to construct the decision rule set are given in Tables 9 and 10. Table 9 describes the partitions utilized (as defined in Equation 15). Both the \vec{v}_N and X_0 components of each partition are specified in terms of the two selection features (i.e., [petal length, petal width]). The $h(X)$ field specifies the the resulting partition function.

The CH region descriptions are provided in Table 10. Since each CH region corresponds to a decision rule, the partition mappings and orientation in this table provide the information needed to construct each rule. The mappings are indexes into the partition list; the indexed partitions are those which enclose the CH region. As we saw in Table 9, each partition is a function which accepts a feature vector

Table 9 Partition Specification for Iris Data.

Designator	\vec{v}_N	\mathbf{X}_0	$\mathbf{h}_i(\mathbf{X})$
\mathbf{h}_1	[0.000, -0.664]	[2.392, 0.603]	$= -0.664(x_4 - 0.603)$
\mathbf{h}_2	[-0.677, 0.000]	[4.789, 1.689]	$= -0.677(x_3 - 4.789)$

Table 10 CH Region to Partition Mapping for Iris Data.

Class	Size (in Members)	Region Center	\mathbf{h}_1	\mathbf{h}_2
Setosa	43	[1.467, 0.249]	1	-
Versicolour	39	[4.221, 1.318]	-1	1
Virginica	37	[5.554, 2.024]	-	-1

(X) as input. In turn, the sign of the function's output determines which *side* of the partition X is on. The orientation value serves to make the function's output *positive* (when the two are multiplied together) for the CH region in question. Since each partition in this problem is univariate, we can substitute a constant for each partition and the symbols $>$ and \leq for orientation values of -1 and 1 , respectively. The result is the decision rule set shown below and (graphically) in Figure 39.

if (petal width ≤ 0.603 cm) then class = Setosa;
 if ((petal width > 0.603 cm) and
 (petal length ≤ 4.789 cm)) then class = Versicolour;
 if (petal length > 4.780 cm) then class = Virginica;

4.9 Adjunct Design Issues

This section addresses extensions to the baseline GRaCCE algorithm. These are primarily intended to optimize the accuracy of the induced rule set.

4.9.1 Resolving Rule Conflicts. A minor disadvantage of the GRaCCE algorithm is that it is possible to evolve CH regions that overlap with each other. This overlap can occur both for regions of the same or different class. Even setting the required purity (δ_{min}) to 100% does not guarantee a region set that is free of

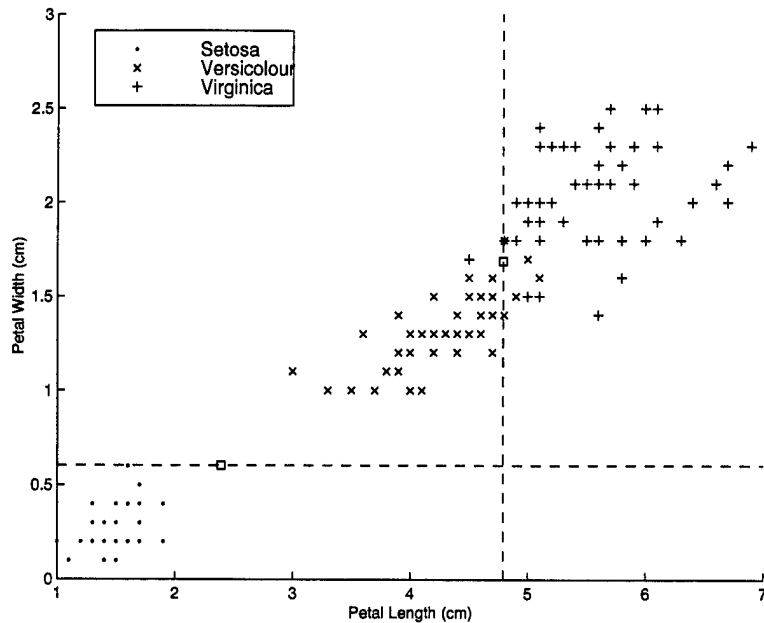


Figure 39 This plot illustrates how the derived partitions are used to separate the classes for the Iris data set.

overlap. This is because the boundary points used to approximate the data set cannot fully represent its actual distribution; in truth, *no* subset of the data can accomplish this. An example of an overlap condition is shown in Figure 40.

Whenever an instance is assigned to two or more regions of differing class, additional logic is needed to resolve the conflict. The approach implemented within GRaCCE is to compile a list of all CH regions that contain the data instance in question. Of these regions, the instance is subsequently assigned to the one with the lowest impurity (with respect to the entire training set). Admittedly, this approach to conflict resolution is somewhat arbitrary; it was chosen because it appeared to produce best results for the data sets tested. Other candidate options that were evaluated include assigning data to the CH region with the largest coverage or to the class with the greatest a priori probability (of those under consideration). There is no guarantee that one of these methods will prove superior to the others for any given data set.

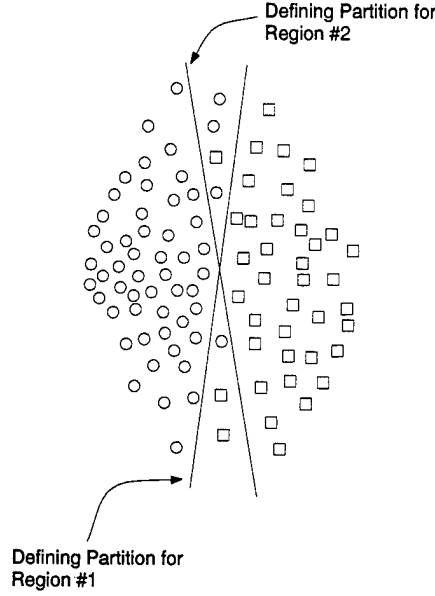


Figure 40 Relaxing the purity parameter (δ_{min}) makes it possible for CH regions of different class to overlap.

4.9.2 Classifying Orphan Data. When the generated rule set is the sole basis for classification, there may be feature vectors which are not assigned to any class. This occurs because the CH regions found by GRaCCE do not always cover the whole data space; as such, orphan data (falling outside these regions) are automatically misclassified. It is possible, however, to reduce the error rate by using criteria in addition to the decision rules to classify data.

A straightforward way to accomplish this is to assign orphan data to the *closest* region. Such an approach is similar in nature to the way a Radial Basis Function (RBF) classifies data (10). The proximity of a given point to the *ith* region is determined using the Mahalanobis distance metric (D_M) (29) computed as

$$D_M(x, R_i) = \sqrt{(x - \mu_i)' \Sigma_i^{-1} (x - \mu_i)} \quad (20)$$

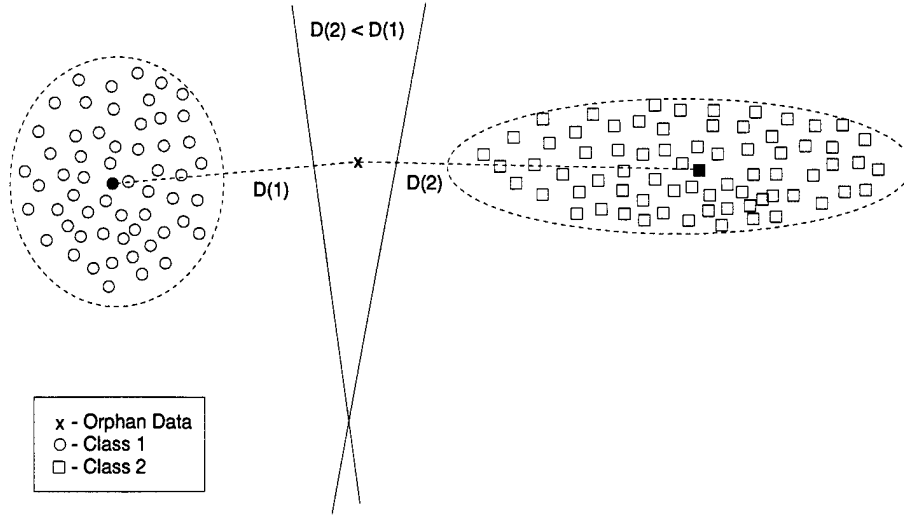


Figure 41 Orphan data can be classified by assigning it to the class of the closest CH region in terms of Mahalanobis distance.

where the mean μ_i and covariance matrix Σ_i are computed from the training data assigned to the region. Unlike Euclidean distance, the Mahalanobis distance metric accounts for the shape of the CH region in its computation, making it a more natural measure of proximity. For example, Figure 41 shows an orphan instance being classified based on its Mahalanobis distance from the center of each CH region. As a result, it is assigned to class 2 due to the hyper-ellipsoidal shape of its CH region even though it is closer (in terms of Euclidean distance) to the center of the class 1 region.

4.10 Toward a Concurrent Version of GRaCCE

Once an algorithm has been successfully implemented, the natural next step is to determine if it can be organized as separate, concurrent tasks. Executing tasks in parallel is attractive due to its potential for accelerating an algorithm's execution. Concurrency is especially important to GRaCCE due to its relatively poor run-time efficiency (as compared to decision tree algorithms). This is due to a number of factors, including its implementation in *MatLab* and the sequential nature of the algorithm. In this section, the design of a concurrent version of GRaCCE (known

as cGRaCCE) is discussed. The material presented here is augmented with an analysis of the algorithm's time complexity Chapter V. In addition, the run-time execution efficiency and scalability of cGRaCCE (on both single and multi-processor configurations) is tested in Chapter VI.

This version of cGRaCCE was designed as a proof of concept demonstration rather than a full-scale implementation. The cGRaCCE system is hosted on the AFIT Beowulf Cluster of personal computers (PCs) described in Appendix E. All work on cGRaCCE was performed as part of Hammack's research (49). Due to time constraints, the project was limited in scope to a single phase of the GRaCCE algorithm. Consequently, the region identification phase was selected as the basis for cGRaCCE because, as shown in Table 11, it is GRaCCE's largest processing bottleneck³. In order to port this code to the Beowulf cluster, it was necessary to rewrite it in C^{++} (from *MatLab* script). In terms of the cGRaCCE parallelization strategy, the following design options were considered:

1. Execute class-specific CH region searches in parallel. With respect to Figure 32, this involves configuring each iteration of the t loop as a separate task (see Figure 42).
2. Run concurrent CH region searches for each boundary point.
3. Within a given CH region search, evaluate all solutions in the population in parallel.

Of the above possibilities, the first option was selected as the basis for cGRaCCE. As Figure 42 illustrates, this option is feasible because all the necessary inputs to each search (such as the set of boundary points and partitions) are generated prior to the start of the phase. In addition, no dependencies exist between searches for each of the m classes. This is because each CH region search only impacts resources

³Although the partition simplification phase has the largest growth rate, its execution is not required.

Table 11 Worst Case Time Complexity of GRaCCE Phases.

Phase	Status	Complexity
Preprocessing - Feature Selection	Optional	$O(nd^2)$
Preprocessing - Winnowing	Mandatory	$O(dn^2d)$
Partition Generation	Mandatory	$O(2(dn)^2 + nd)$
Data Set Approximation	Mandatory	$O(n \log(n))$
Region Identification	Mandatory	$O(qn^2d^{3/2})$
Region Refinement	Mandatory	$O(dn \log(n))$
Partition Simplification	Optional	$O(d^4n^2 \log(n))$

related to its target class (such as boundary point assignments). As a result, the region identification phase for each class can be treated as a series of independent tasks, each executed on its own processor. This approach has the *potential* of yielding a factor of m speedup relative to the sequential version of GRaCCE. Both of the remaining options would have generated far more tasks than available processors. In addition, the second option would have required additional post-processing logic to select the minimal set of regions that cover all boundary points for each class. Because this task is equivalent to the NP-complete Set Covering problem (4), implementing it to would adversely increase the existing algorithm's time complexity. Given these considerations, the first option was judged as most compatible with both the Beowulf configuration and the limited scope of this effort. A more detailed description of the cGRaCCE design was provided by Hammack (49).

Since the current version of cGRaCCE only implements a single phase, addressing the remaining phases is a top priority for researchers using this system in the future. Nonetheless, parallelizing the rest of the system should be fairly straightforward. Perhaps the best candidates for parallelization are the preprocessing (winnowing) and partition generation phases⁴. In both these cases, the tasks that comprise each phase can be sub-divided into equal sized slices and assigned to

⁴This applies only to the generation of the initial set of boundary point pairs and their associated partitions.

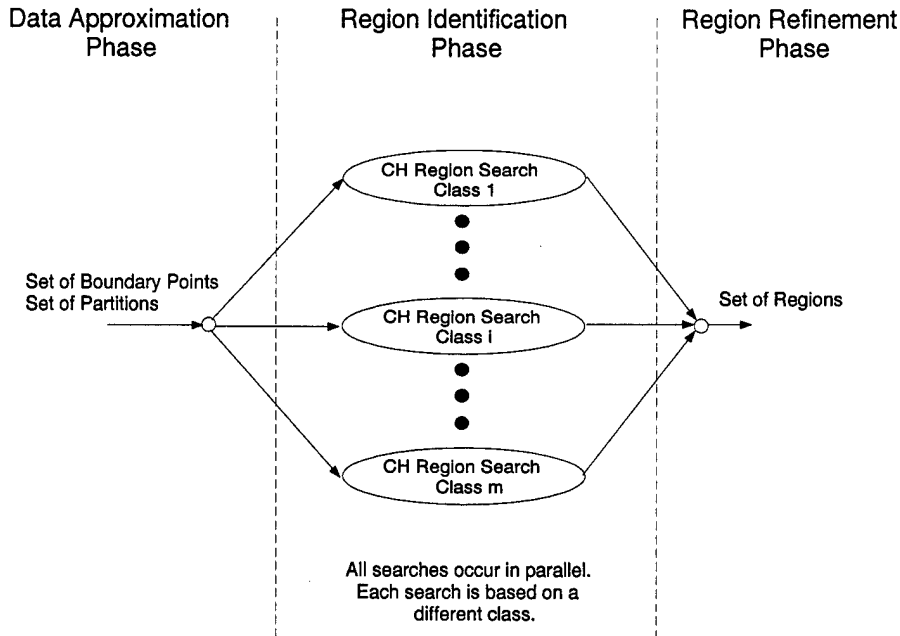


Figure 42 Flowchart for cGRaCCE Tasks.

separate processors for concurrent execution. In turn, the global set of data required for each task is shared among all processors in the cluster. As Figure 43 illustrates, the partition generation task can be divided among multiple processors if each processor is assigned a slice of the boundary point pair set, while all processors share the data set as a whole. Table 12 describes the division of labor scheme for each candidate task.

Note that even with the above approach, it is still unnecessary (or impractical) to parallelize some aspects of GRaCCE. For example, the data set approximation phase is efficient enough that making it concurrent would yield little benefit. In addition, parallelizing the iterative partition generation algorithm (refer to Figure 29) is impractical since it is essential that each boundary point be tested on an up-to-date version of Γ . Similarly, designing a concurrent region refinement or partition simplification phase is difficult since a change to one region (or partition) may affect the effectiveness of others in the set. This forces changes to R and Γ to be made in a

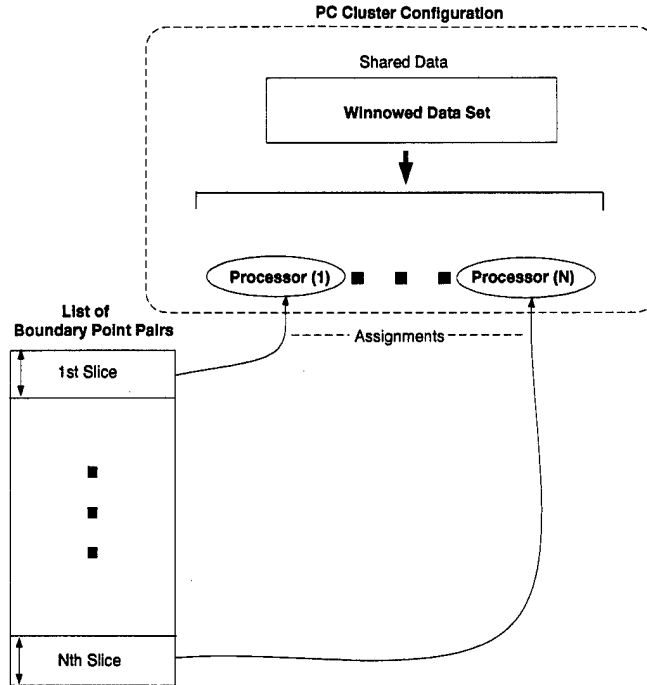


Figure 43 Parallelization Scheme for Extending cGRaCCE.

Table 12 Scheme for Parallelizing Candidate GRaCCE Tasks.

Task	Divide	Share	Generated Output
Winnowing	Data Set	Data Set	Winnowed Data Set (WDS)
Boundary Point ID	WDS	WDS	Boundary Point Pairs (BPP)
Partition Generation	BPP	WDS	Partition Set

serial (rather than parallel) fashion. This is unfortunate as the partition simplification phase has the worst time complexity. Given this, it may be prudent to explore replacements for this functionality in the future.

4.11 Summary

This chapter has provided a comprehensive description of how GRaCCE constructs classification rule sets. We conclude by summarizing those characteristics that make GRaCCE fundamentally *different* from other rule induction methods. The first important distinction is the way class decision boundaries are derived. The

point has repeatedly been made that GRaCCE finds class partitions that approximate the natural decision surface, similar approaches are also used by methods such as piecewise linear classifiers (123). What makes the GRaCCE approach unique, however, is that it first removes noisy instances from the training set which can obscure the *true* decision boundary. In addition, the derived partitions approximate the decision boundary both on a global (per class) and local (per boundary point pair) basis. Lastly, GRaCCE uses an iterative partition generation technique to insure that the partition set is sufficient to enclose each boundary point.

Because of the rigor of the above process, GRaCCE starts out with a large pool of good building blocks for the construction of CH regions. The second distinction is predicated on how the system uses this resource. Unlike decision tree methods, which select a locally optimal partition to split each data subset, GRaCCE uses evolutionary search to evolve a CH region from the partitions in its pool. This gives it an advantage in that it can evaluate combinations of partitions, rather than just one at a time. While other methods possess this capability (such as classifier systems), the key difference here is the types of partitions considered. Recall from Chapter II that classifier systems use partitions based on a single, discrete feature. As a result, these partitions may not approximate the class decision boundaries found in “real-world” data as successfully as those generated by GRaCCE. These points support the argument that GRaCCE represents a unique and innovative rule induction method that combines the best elements of other approaches. In subsequent chapters, the theoretical and empirical support for this assertion are explored.

V. *Properties of GRaCCE*

This chapter explores the properties of GRaCCE from a theoretical and algorithmic perspective. With regard to the former, a number of theorems are proven which show GRaCCE generates a set of partitions that approximate the natural class decision boundaries. It is further demonstrated that these partitions are sufficient to construct a complete set of decision rules to classify the data. The latter half of the chapter focuses on deriving the time complexity of the GRaCCE algorithm (for both its sequential and concurrent forms). Throughout the chapter, the mathematical concepts that support the discussion are carefully developed.

5.1 *Theoretical Foundations*

This section provides a brief review of Bayesian decision theory as an introduction to the data preparation methods employed by GRaCCE. Using references to existing literature, it is proved (through a series of theorems) that these methods lead to the generation of partitions that are piecewise approximations of the Bayes optimal decision boundary. It is further proved that it is theoretically possible to find combinations of these partitions that form class homogeneous regions that optimize a given goodness criteria.

5.1.1 The Bayes Theorem and Related Concepts. Much of pattern recognition theory is built on a foundation of statistical mathematics; in turn, a large part of the corresponding statistical theory is based on the work of Bayes. This section examines the fundamentals of Bayesian decision theory.

The “centerpiece” of decision theory is the Bayes Theorem. Consider a problem with m classes and a random variable x . Under these conditions, Bayes theorem states that the a posteriori probability of class ω_i given x is given by

$$p(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)}, i = 1..m \quad (21)$$

where $p(x|\omega_i)$ is the conditional probability density function (PDF) of x given ω_i , $P(\omega_i)$ is the a priori probability of class ω_i . The class independent PDF for x , $p(x)$, is defined by

$$p(x) = \sum_{i=1}^m p(x|\omega_i)P(\omega_i) \quad (22)$$

These statistics can be estimated from the training data for any pattern recognition problem (29). As a result, it is possible to use these statistics to make a decision regarding the class of x with a minimum of error. For a two class problem this can be achieved using the *Bayes decision rule*. Let x be an observation vector belonging to either ω_1 or ω_2 . According to the Bayes decision rule, we can determine the most likely class for x based on the criteria:

$$\begin{aligned} x &\in \omega_1 \text{ if } p(\omega_1|x) > p(\omega_2|x) \\ x &\in \omega_2 \text{ if } p(\omega_2|x) > p(\omega_1|x) \end{aligned}$$

Thus, we select class ω_1 when $p(\omega_1|x) > p(\omega_2|x)$. For a multi-class problem, we can generalize this class selection method using Equation 23. Thus, the decision boundary between two classes is reached when $p(\omega_1|x) = p(\omega_2|x)$. The discriminant function (h) that describes this decision boundary is defined by Equation 24.

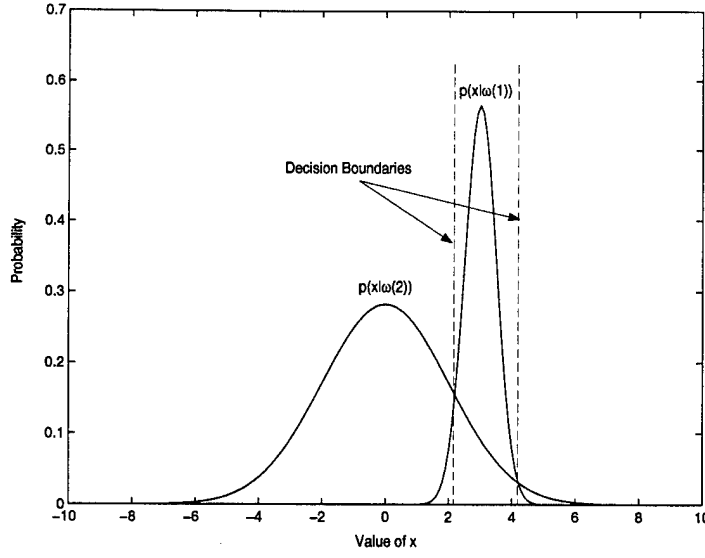


Figure 44 Relationship of PDFs to Decision Boundaries.

$$\text{Select } \omega_k \text{ such that } p(\omega_k|x) = \max_i \{p(\omega_i|x)\} \quad (23)$$

$$h(x) = -\ln(p(x|\omega_1)) + \ln(p(x|\omega_2)) \begin{matrix} > \\ < \end{matrix} \ln \frac{P(\omega_1)}{P(\omega_2)} \quad (24)$$

If equal prior probabilities are assumed, the relationship between class distributions and the decision boundaries are illustrated by example in Figures 44 and 45.

5.1.2 The Bayes Error. If we have complete knowledge of a given data set's characteristics, it is possible to derive the minimum possible error by integrating over those regions where a given class distribution is in error. For a two class problem, the conditional error (r) for any given x due to the Bayes decision rule is given by

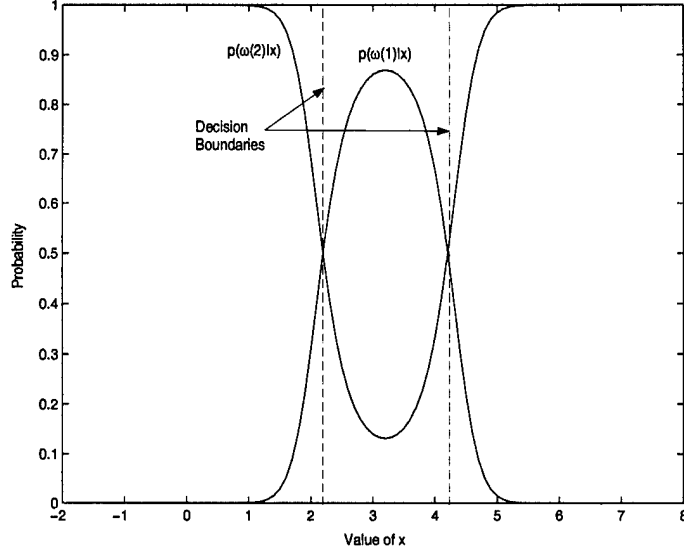


Figure 45 Relationship of A Posteriori Probabilities to Decision Boundaries.

$$r(x) = \min\{p(x|\omega_1)P(\omega_1), p(x|\omega_2)P(\omega_2)\} \quad (25)$$

Since the Bayes decision rule is optimal, r represents the minimum possible error at x . Thus by taking the expected value of r (as in Equation 26), we can compute the minimum error for the problem; this is known as the *Bayes error* (ϵ_B). In particular, if L_i is the region where the i th class has the highest posterior probability, to find the Bayes error for the i th class, we must integrate over the region $L_{\neq i}$. Equation 27 computes the Bayes error for a multi-class problem.

$$\begin{aligned} \epsilon_B &= E[r(x)] \\ &= \int r(x)p(x)dx \\ &= P(\omega_1) \int_{L_2} p(x|\omega_1)dx + P(\omega_2) \int_{L_1} p(x|\omega_2)dx \end{aligned} \quad (26)$$

$$\epsilon_B = \sum_{i=1}^m 1 - P(\omega_i) \int_{L_i} p(x|\omega_i) dx \quad (27)$$

For a given data set, the Bayes error sets the fundamental limit on classification performance (37). As a result, the Bayes error is an extremely important metric to use in assessing the performance of a classifier algorithm. From a practical standpoint, however, computing the Bayes error is not always possible. In some cases, the class likelihoods and priors are unknown; in others, these are known, but the result is difficult to compute numerically (141).

When such situations arise, the solution is to compute an estimate for the bounds of the Bayes error. One of the most basic Bayes error estimates is the Bhattacharyya distance (ρ) (37). By using

$$\min\{a, b\} \leq \sqrt{ab} \quad (28)$$

we can transform Equation 26 into

$$\begin{aligned} \epsilon_B &\leq \sqrt{P(\omega_1)}\sqrt{P(\omega_2)} \int_{-\infty}^{+\infty} \sqrt{p(x|\omega_1)p(x|\omega_2)} dx \\ &\leq \sqrt{P(\omega_1)}\sqrt{P(\omega_2)} \exp(-\rho) \end{aligned} \quad (29)$$

which provides an upper limit on ϵ_B . The ρ term is defined by

$$\rho = \frac{1}{8}(\mu_2 - \mu_1)' \left(\frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\mu_2 - \mu_1) + \frac{1}{2} \ln \frac{|\frac{\Sigma_1 + \Sigma_2}{2}|}{\sqrt{|\Sigma_1||\Sigma_2|}}. \quad (30)$$

Using ρ , the bounds on ϵ_B from above and below can then be calculated with the following equation:

$$\frac{1}{2} \left(1 - \sqrt{1 - 4P(\omega_1)P(\omega_2)\exp(-2\rho)} \right) \leq \epsilon_B \leq \exp(-\rho)\sqrt{P(\omega_1)P(\omega_2)}. \quad (31)$$

While it provides relatively tight bounds on the Bayes error, the Bhattacharyya distance is much less accurate when the data does not conform to a Gaussian distribution. As a result, non-parametric methods of estimating the Bayes error may be preferable when little is known about the underlying distribution of the data.

5.1.3 Properties of the kNN Algorithm. The kNN algorithm (10) is one of the most fundamental non-parametric methods for estimating the Bayes error. Starting with Cover and Hart (20), a number of researchers have proved that as the number of samples gets larger, the error probability for the kNN algorithm asymptotically (*) bounds the Bayes error probability. In particular, when an infinite number of samples is available the conditional error given x is

$$r(x)^* = 2p(\omega_1|x)p(\omega_2|x) \quad (32)$$

Using Equation 25, it can be shown that this relation is equivalent to $2\epsilon_B$ (37). Thus, the error produced by the kNN algorithm is said to provide an upper bound of twice the Bayes error. More precisely, Fukunaga (37) shows the following relationship between the value of k and the kNN bound on the Bayes error:

$$\frac{1}{2}\epsilon_B^* \leq \epsilon_{2NN}^* \leq \epsilon_{4NN}^* \leq \dots \leq \epsilon_B^* \leq \dots \leq \epsilon_{3NN}^* \leq \epsilon_{1NN}^* \leq 2\epsilon_B^* \quad (33)$$

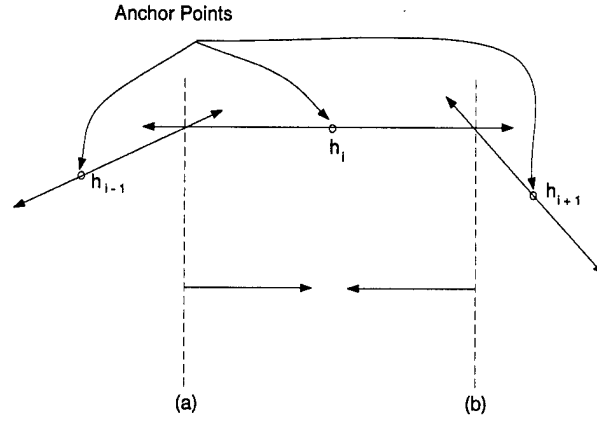
While this bound is of theoretical importance, an upper bound of $2\epsilon_B$ is still a relatively imprecise error estimate. Further investigation by Devijver and Kittler (27) revealed additional properties relating to the edited kNN procedure (refer to Section 3.7). Given that r_{kNN}^* is the asymptotic error probability for the kNN and $r(x)$ is the Bayes optimal conditional error, they proved the following is true of repeated kNN editing:

$$\lim_{j^+} r_{kNN}^* = r(x) \quad (34)$$

This means that as the number of edits (j) increases, the conditional error converges to the Bayes optimal conditional error. Thus, the edited kNN procedure can yield a tighter approximation of $2\epsilon_B$ than the standard kNN algorithm. In addition, Devijver and Kittler assert that the final edited data set contains a fraction $(1-2\epsilon_B)$ of the number of samples before editing. As a result, the kNN editing procedure leaves the remaining clusters densely populated and the decision boundaries fairly well defined for reasonable problems (27). These important findings are central to the theorems and proofs that follow in Section 5.2.

5.1.4 Relevant Definitions. In addition to the definitions for boundary points (Definition 1) and boundary point pairs (Definition 2) presented in Section 4.3, the following definition is also relevant:

Definition 6 - A contiguous sequence of linear partitions (H). Let H be a sequence of n adjacent partitions separating two arbitrary classes (ω_1 and ω_2) over an interval $[a, b]$ such that:



The interval $[a,b]$ shrinks as $n \Rightarrow \infty$, such that $\text{distance}(a,b) < \epsilon$

Figure 46 Piecewise Sequence of Linear Partitions.

$$H(x) = \left\{ \begin{array}{ll} h_1(x) & \text{for: } a \leq x < x_1, \\ h_2(x) & \text{for: } x_1 \leq x < x_2, \\ \dots & \dots \\ h_n(x) & \text{for: } x_{n-1} \leq x < x_n = b. \end{array} \right\}$$

when $h_i(x_i) = h_{i+1}(x_i)$ for $i = 1, \dots, n-1$ and $a < x_1 < x_2 < \dots < x_n = b$.

Given Definition 6, Figure 46 illustrates how the interval over which each partition is active shrinks as the number of partitions in the sequence increases. This concept becomes significant when Theorem 3 is discussed in the next section.

5.2 GRaCCE-Related Theorems and Discussion

In this section, three theorems related to the properties of GRaCCE are introduced and proved. The practical implications of these theorems, with regard to rule induction, are then discussed.

5.2.1 Theorem Development.

Theorem 1 *Assume: that each class has a continuous PDF. Given that the NN editing procedure converges to the Bayes optimal error (27), the Bayes decision boundary must lie between a boundary point pair (b_1, b_2) .*

Proof.

1. Since each edited data set supports a Bayes optimal error condition, then:
 $p(b_1|\omega_1) > p(b_1|\omega_2)$ and $p(b_2|\omega_1) < p(b_2|\omega_2)$.
2. Given that x_D is a point on the Bayes decision surface, $p(x_D|\omega_1) = p(x_D|\omega_2)$.
3. Let $h(x)$ be defined as: $h(x) = p(x|\omega_1) - p(x|\omega_2)$, then $h(b_1) > 0$ and $h(b_2) < 0$.
4. By Theorem 4.18 from Apostol (5, page 80), $h(x)$ is continuous at $h(x_D)$.
5. By Bolozano (5, page 85), $x_D \in (b_1, b_2)$. *QED.*

Given Theorem 1, it is desirable to compute a local approximation of the Bayes decision boundary between b_1 and b_2 .

Theorem 2 *A hyper-plane tangent to the Bayes decision surface separating a boundary point pair (b_1, b_2) can be constructed from the neighborhood (of like-class) surrounding each boundary point.*

Proof.

1. A hyper-plane is defined by a point in space (x) through which it passes and a vector normal to the hyper-plane (\vec{v}_N).
2. Faux and Pratt (32) derived Equation 35 for computing \vec{v}_N normal to the decision boundary at X_0 (known as the anchor point). While this equation is useful, Faux and Pratt did not present a method for computing X_0 .
3. Based on (32), Lee and Landgrebe (81) derived a rather complex method for finding X_0 .

4. Both of the above methods assume a Gaussian distribution for the data of each class with parameters derived using the maximum likelihood estimator (MLE). Since the MLE is consistent (59), the parameter estimates converge to their actual values (i.e., $\hat{\mu}_i \rightarrow \mu_i$, $\hat{\Sigma}_i \rightarrow \Sigma_i$) as the neighborhood around each boundary point becomes increasingly dense.
5. As a result, the derived hyper-plane (h) becomes a local approximation to the Bayes decision surface in the region between b_1 and b_2 . *QED*.

$$\vec{v}_N = \nabla h(X)|_{X=X_0} = (\Sigma_1^{-1} - \Sigma_2^{-1})X_0 + (\Sigma_2^{-1}\mu_1 - \Sigma_1^{-1}\mu_2) \quad (35)$$

Theorem 3 *The composite sequence of piecewise linear partitions separating classes ω_1 and ω_2 (as defined in Theorem 2) converges to the Bayes decision surface as the number of boundary point pairs increases.*

Proof.

1. Let h be the Bayes decision surface; and H be the composite decision surface as described by Definition 6. Figure 47 depicts such a sequence.
2. Given the conditions in Theorem 2 and $n \rightarrow \infty$, then by the Cauchy uniform convergence theorem (5, page 220), $|H(x) - h(x)| < \epsilon$, $\forall x$ on the Bayes decision surface for each fixed $\epsilon > 0$. *QED*¹.

Theorem 4 *Given that GRaCCE utilizes the algorithm described in Figure 29, then sufficient partitions are generated to enclose every boundary point in a class homogeneous (CH) region.*

Proof. Recall that the iterative partition generation algorithm presented in Section 4.3 (see Figure 29) sequentially processes every instance (b) in the boundary

¹Ruck (110) provides a similar proof that shows MLPs approximate the Bayes optimal discriminant function.

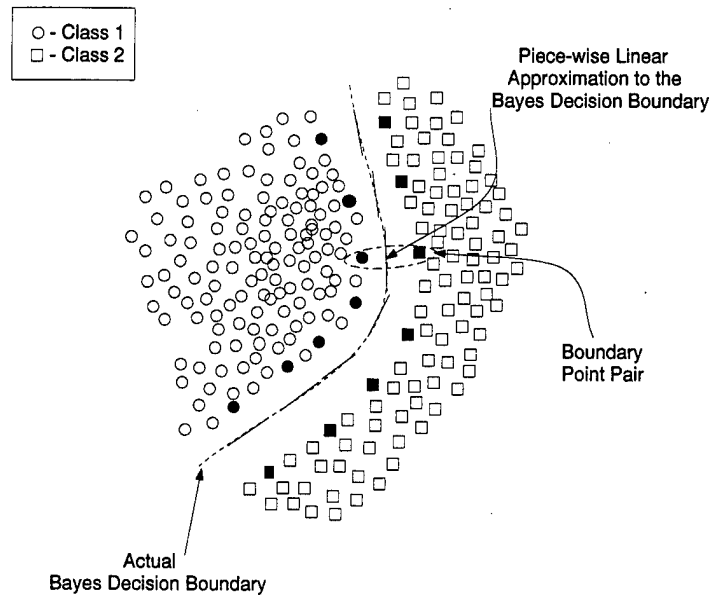


Figure 47 Combination of Local Partitions Approximate the Bayes Decision Surface.

point list (Λ). Each b is tested to see if it can be enclosed in a CH region using the current partition set (Γ). If this condition is not satisfied, the necessary additional partitions and boundary points are generated until it is enclosed. These items are appended to their respective sets (Λ , Γ). The algorithm continues until all boundary points in Λ have been processed. Therefore, by contradiction, it is impossible to have a boundary point which does not have an associated CH region. *QED*.

5.2.2 Discussion. The relative goodness of the rules generated by GRaCCE are predicated on the quality of the conditions upon which they are based. In this case, the conditions for each rule correspond to specific inter-class partitions. The purpose of the theorems presented here is to establish a formal basis for evaluating these partitions. Since using the Bayes decision surface minimizes misclassification error, partitions that approximate it have inherent goodness and thus provide a solid foundation for constructing decision rules.

The theorems prove that GRaCCE produces partitions that are *piecewise approximations* to the Bayes decision boundary (as illustrated by Figure 47). In par-

ticular, Theorems 1 and 2 show that GRaCCE is able to derive local approximations to the Bayes decision boundary for each boundary point pair. This proof is used by Theorem 3 to show that the composite of the piecewise partition set converges to the Bayes decision boundary separating two classes (see Figure 46 for a pictorial representation of this process). Given a set of such partitions, the critical question to ask is: can GRaCCE construct optimal CH regions given these partitions? Recall from Section 4.5, a CH region is considered optimal if it minimizes the objective function (Φ) for a given purity constraint (δ_{min}).

Since GRaCCE uses a genetic algorithm (GA) consistent with Bäck's definition (6, page 122), his GA convergence theorem (stated in Theorem 5) also applies. The first condition of the convergence theorem is satisfied when GRaCCE employs an elitist reproduction strategy; the user may select this option by varying the Probability of Replacement parameter in the GA Menu (see Appendix B for details). The second condition (reachability) is satisfied as a result of Theorem 4, which shows that sufficient partitions exist to form CH regions for each boundary point. Since the GA employs both mutation and recombination, GRaCCE is capable of optimizing Φ with a countably infinite number of generations.

Theorem 5 - GA Convergence Theorem. *Let a Genetic Algorithm (GA) fulfill the following conditions:*

(1) *The population sequence $P(0), P(1), \dots$ is monotone, i.e., $\forall t$:*

$$\begin{aligned} & \min\{\Phi(\vec{a}(t+1)) | \vec{a}(t+1) \in P(t+1)\} \\ & \geq \min\{\Phi(\vec{a}(t)) | \vec{a}(t) \in P(t)\}, \text{ and} \end{aligned}$$

(2) *$\forall \vec{a}$, the optimal solution $\vec{a}^* \in I$ is reachable from \vec{a} by means of mutation and recombination.*

Then:

$$\mathcal{P}\{\lim_{t \rightarrow \infty} \vec{a}^* \in P(t)\} = 1.$$

Proof. Refer to Bäck's proof (6, page 129). *QED.*

5.3 Search Space Considerations

The induction of optimal decision trees is complicated by two factors. The first is that it is an NP-complete problem (63); as a result, it is necessary to explore all possible tree permutations to determine if an optimum has been achieved. The second is the intractability of accomplishing such a search considering the sheer number of possibilities involved. Consider that given a data set containing n instances of dimension d , there can be $2 \times \sum_{k=0}^d \binom{n-1}{k}$ oblique splits if $n > d$ (140); each split is a hyper-plane that divides the search space into two non-overlapping parts.

It is important to remember that these are estimates of the size of the partition pool; they do not represent the number of ways the partitions can be combined to produce a decision tree. Equation 36 computes the total number of unique combinations where k is the size of the partition pool and r is the number of partitions used in the final tree. This yields an astronomical number of combinations even for a relatively small problem.

$$C(k, r) = \binom{k}{r} = \frac{k!}{(k-r)!r!} \quad (36)$$

Due to the intractability of the problem, exhaustive search is rarely a viable option for decision tree induction. As a consequence, most of these algorithms are greedy in nature (11; 105; 94). While these strategies are effective in finding locally optimal solutions for each decision node, the results they produce are suboptimal in terms of the final tree structure (45). In addition, the search for a partition is based on minimizing some metric (such as impurity). As discussed earlier, these metrics

make assumptions about the data landscape that may not be true. Thus, there is no guarantee that these metrics yield partitions that approximate the Bayes decision surface.

In contrast to these greedy strategies, GRaCCE preprocesses the search space to identify a pool of partitions that approximates the Bayes decision surface between boundary point pairs. While the size of the GRaCCE partition pool can theoretically be $C(n, 2)$, in practice it is much more tractable ($k \ll n$). For problems involving two or more classes, the partition pool is even smaller since only those partitions related to the target class are utilized in a search. Thus, compared to the decision tree algorithms, the search space used by GRaCCE is smaller and it is populated with higher quality building blocks.

Given the previous discussion, it is feasible to employ genetic search to locate CH regions using the generated partition pool. Unlike the greedy algorithms used in decision tree algorithms (94; 105; 111), GAs are capable of finding CH regions that maximize the criteria in Φ . This assertion is supported by researchers in combinatorial optimization who observed that randomized search usually succeeds when the search space holds an abundance of good solutions (48). Theorems 1 thru 4 show that GRaCCE fits this description by utilizing a pool of good partitions sufficient for enclosing each boundary point in a CH region. While the No Free Lunch (NFL) theorem (152) implies that no algorithm outperforms another under all conditions, it is reasonable to expect that GRaCCE should routinely find solutions better (in at least one of the optimization criteria) than those yielded by the oblique decision tree algorithms.

For axis parallel splits, the number of potential partitions is a much lower, but still significant $n \times d$ (94). Since the search space is much smaller, finding an optimal solution using a deterministic search becomes a much more attractive alternative. In addition, GA-based Classifier Systems (41; 60; 127), have also been applied to these problems. Given this, the desirability of employing GRaCCE would seem to

fade. Recall from Section 2.2.2, however, that the smaller search space is due to constraints on the form of the rule set. Thus, if these constraints do not match the actual structure of the data (i.e., the classes cannot be separated well using axis-parallel hyper-planes), the generated decision rule set can be more complex than necessary.

5.4 *Fitness Landscape Considerations*

The concept of a fitness landscape (70) refers to the mapping from the genomes of a population of individuals to their fitness, and a visualization of that mapping (also refer to Appendix A). An analysis of the structure of fitness landscapes is of both theoretical and practical interest in that the difficulty of the problem is closely related to the ruggedness of the landscape's structure. Thus, a smooth fitness landscape tends to denote an easy problem while a rough landscape indicates a difficult one. Because landscapes for binary GA problems are difficult to visualize, a number of researchers have sought to characterize these landscapes by measuring how well correlated they are (148). For example, in a smooth landscape, one would expect genetically similar individuals (neighbors) to have similar levels of fitness; the converse is true in a very rugged landscape. One technique used to assess landscape correlation is the adaptive walk (72). This involves incrementally changing the genotype in some manner that is *expected* to improve fitness, and measuring the degree to which fitness changes; each distinct genotype change is called a *step*.

We use a variation of the adaptive walk technique to characterize the fitness landscape of GRaCCE. In this case, the fitness for every adaptive step taken by a greedy search algorithm is measured. This algorithm (described by the pseudocode in Figure 33) searches for an initial CH region solution to initialize the GA population with. The algorithm begins by using every partition to define the CH region and then tries to minimize the objective function (see Equation 18) by sequentially testing the fitness contribution of each partition. Partitions that can be discarded without

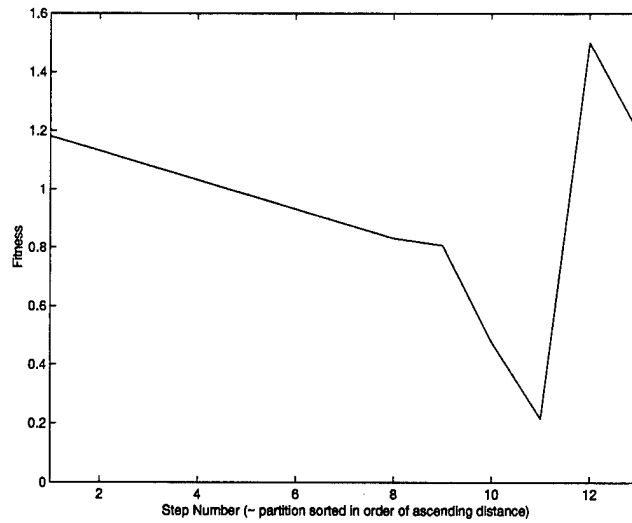


Figure 48 Results of Greedy, Adaptive Walk for the Wine Data Set.

worsening fitness are eliminated from the solution. The order of testing is determined by distance to the focal (boundary) point of each search. The closer a partition is to the focal point, the sooner it is tested. Thus each test of the greedy solution is considered a step.

Figure 48 shows a slice of the landscape observed for one such (greedy) adaptive walk for the Wine data set (refer to Table 15). As can be seen from the plot, fitness starts out high and improves steadily until the 11th partition is eliminated. At this point, the purity constraint in the objective function is violated and the fitness worsens considerably. In short, this example implies that regions of the landscape where solutions possess *essential* partitions (to enclose a given boundary point) have a relatively smooth landscape, leading to a “valley” of near optimal solutions. On the other side of the valley, “cliffs” arise when solutions shed these essential partitions. Thus, the purpose of the greedy algorithm is to find solutions near this valley so that the GA can commence a more focused local search. While each data set is different, the Wine data set has classes that can be characterized by unimodal, Gaussian distributions. Intuitively, we would expect the fitness landscapes of data sets with similar characteristics to behave likewise.

5.5 Time Complexity Analysis

This section derives the GRaCCE algorithm's run-time execution growth rate. Such an estimate provides a theoretical basis for comparing the system's run-time efficiency with those of other rule induction methods. Due to its importance in the algorithm, the Region Identification (RI) phase (discussed in Section 4.5) is the exclusive focus of this analysis. Compared to the other phases, the time complexity of the RI phase is most heavily dependent on the specific structure of the data set being processed. Accordingly, we begin by constructing a model of the data set based on some fundamental assumptions about its structure. Using this model, a complexity estimate is derived for the RI phase. Next, this model is updated for the concurrent version of the algorithm proposed in Section 4.10. Lastly, the derived complexities are compared to those of decision tree algorithms.

5.5.1 Data-Related Assumptions. The time complexity of the GRaCCE algorithm is closely linked to the structural characteristics of the data set being processed. In particular, it is most dependent on the number of partitions and boundary points generated from a given data set. The more boundary points (and corresponding partitions), the greater the processing load during the region identification phase. While the specifics of this relationship will be explained in Section 5.5.2, the data set attributes that affect these factors are discussed here.

Given a particular data set with n training instances, n_{bp} boundary points are yielded by the GRaCCE algorithm. In a typical data set, $n_{bp} \ll n$. Although there is no set formula for determining this relationship, the actual number of boundary points is influenced by the following data set characteristics:

- Number of training instances.
- Number of classes in the training data.
- Number of modes per class and their relative position.

Table 13 Complexity Metrics for Example Data Sets.

Figure	Distribution	n	m	Modes	$ \Lambda $	$ \Gamma $
49	Gaussian	1000	2	2	4	4
50	Uniform	1000	2	2	9	7
51	Gaussian	1000	2	20	14	10
53	Gaussian	1000	2	25	105	66
52	Gaussian	1000	5	25	63	49

- The type of data distribution (i.e., Gaussian, uniform, etc).

The extent to which these factors affect partition generation is reflected by the cases described in Table 13 and shown in Figures 49 through 52. Despite the fact that all cases have the same amount of data, note that there is a substantial difference in the number of generated partitions. In Figure 49 (the simplest case), only a few partitions are generated due to the unimodal/Gaussian nature of the data set. In contrast, the uniform distribution of data in Figure 50 causes many more boundary point pairs (and corresponding partitions) to be produced. A comparison of Figures 51 and 53 reveals how a shift in the spatial orientation of modes can affect the number of partitions generated. Lastly, Figure 52 demonstrates that increasing the number of classes does not automatically translate into more partitions. The moral of the story is that you cannot judge the structural complexity of a data set by its size alone.

This variation raises the question: is there an easy way to estimate a data set's complexity based on its size (n)? If one is interested in a precise estimate, then the answer to this question is clearly no. As the previous cases illustrated, the difficulty in prediction is caused by the way one characteristic can offset another. In addition, every data set has a unique distribution which any randomly selected subset of the data should conform to. This situation makes it necessary to create a general purpose model to describe the data. Once such a model is defined, more reliable estimates of the algorithm's complexity can be obtained. Since GRaCCE

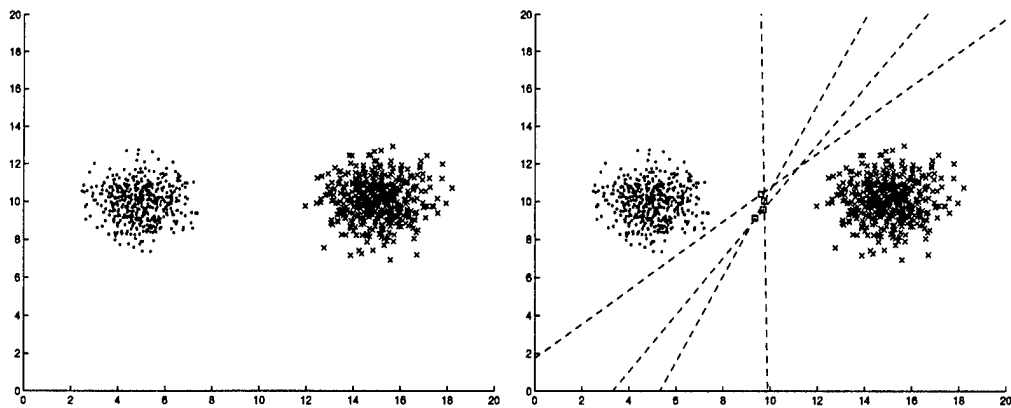


Figure 49 Two Class Unimodal - Gaussian Distribution.

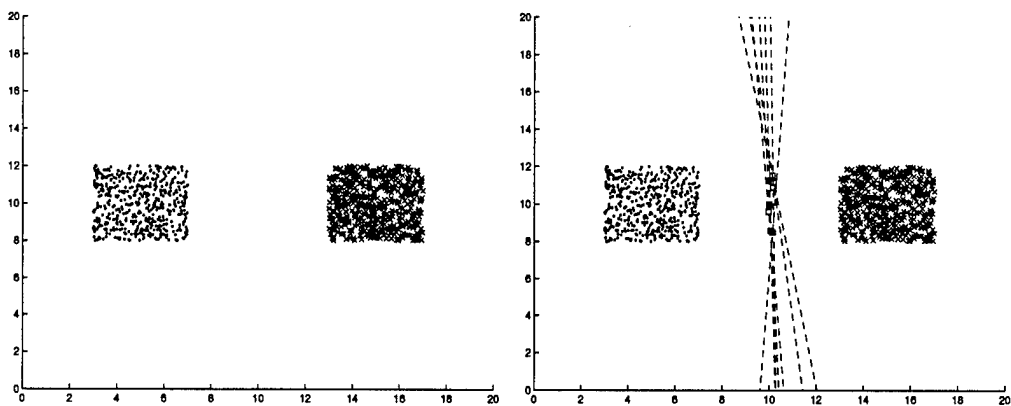


Figure 50 Two Class Unimodal - Uniform Distribution.

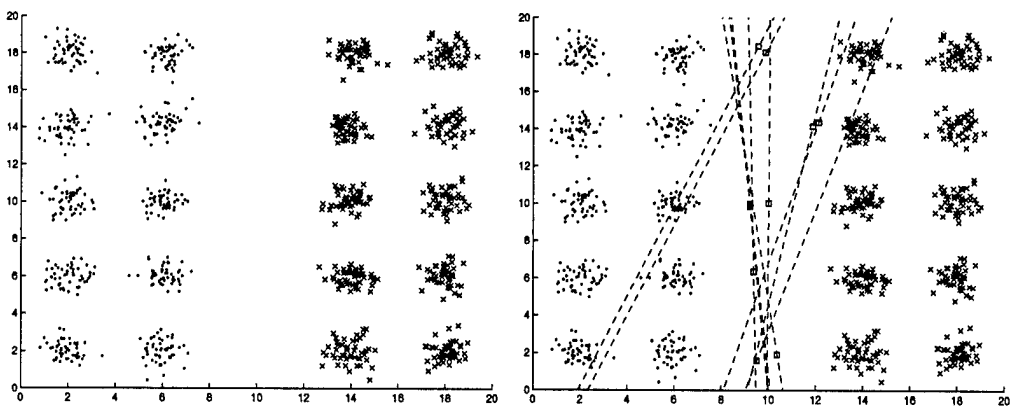


Figure 51 Two Class Multimodal - Symmetric.

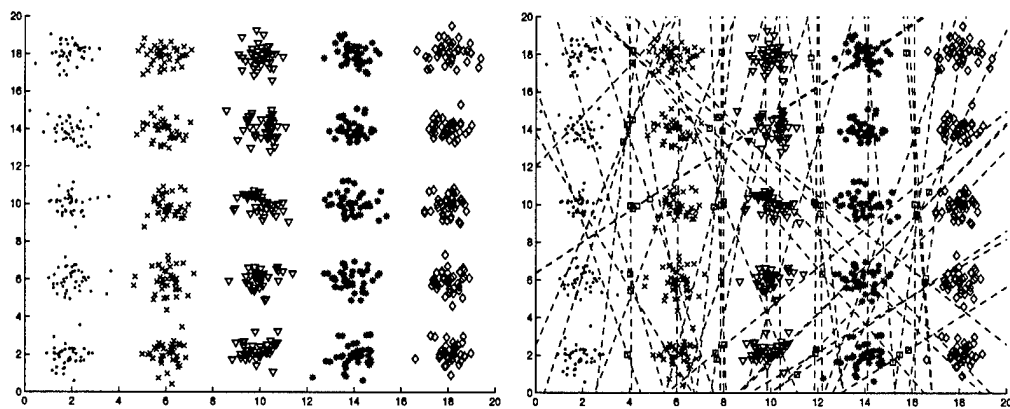


Figure 52 Five Class Multimodal - Interleaved.

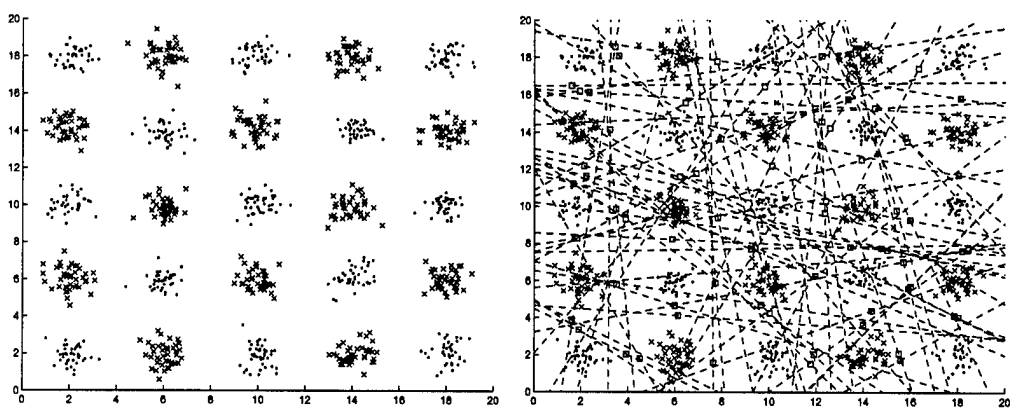


Figure 53 Two Class Multimodal - Checkerboard.

is specifically engineered to handle highly multi-modal data sets, we create a data model for our analysis based on the following assumptions:

- Each mode is its own class (in the context of this discussion, class and mode are synonymous).
- Each mode has a Gaussian distribution.
- All modes have the same number of instances.
- The number of instances per class increases exponentially with the data set's dimensionality.

Given these assumptions, our data model has two parameters: number of classes (m) and dimensionality (d); therefore, this model is denoted as $\Omega(m, d)$.

The next step in the analysis is to estimate the *worst case* number of generated partitions ($|\Gamma|$) given our data model. This is a critical step since the number of boundary points is closely related to the number of partitions. Recall GRaCCE generates two types of partitions: global and local. While $m(m - 1)/2$ global partitions are generated (one for each pair of classes), estimating the number of local partitions is a far more difficult task.

To develop this estimate, it is helpful to represent the problem as a graph of dimension d where each mode is treated as a distinct vertex. In this representation, each edge corresponds to a boundary point pair consistent with Definition 2. Note that this definition precludes the use of a fully connected graph which would result in $C(m, 2)$ edges. Thus, we must construct the graph such that each new vertex adds the maximum number of edges to the graph without causing existing edges to be invalidated. Within these constraints, the maximum number of edges is d provided that all edges are of equal length (ensuring that each vertex is equidistant to all its neighbors). Figure 54 illustrates such a case for two dimensions. The upper limit on the maximum number of edges in such a graph is given by

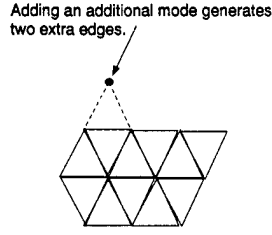


Figure 54 Data set configuration for worst case complexity.

$$\text{Edges} \approx dm \geq \lfloor \left(\frac{m-1}{d+1} \right) \rfloor + d(m-d) \quad (37)$$

when $m > d + 1$. Since each edge is a potential class partition, then $|\Gamma|$ grows at a worst case rate of

$$O_{\Gamma} = dm + \frac{m(m-1)}{2} \quad (38)$$

when both the global and local partitions are accounted for. It is further assumed that the partition and boundary point sets grow at approximately the same rate because each partition has an associated boundary point pair.

5.5.2 Derivation of Sequential Time Complexity. The time complexity for the sequential version of the RI phase can now be estimated using the previous assumptions. The necessary calculations in this section are based on Figures 55 and 56 which contain meta-level versions of the RI phase and objective function, respectively. While not a detailed representation, this pseudocode includes the generic operations that directly impact the algorithm's overall complexity. The analysis starts with the GA in the Figure 55 pseudocode and proceeds outward to yield the desired estimate.

```

for  $t = 1$  to  $m$  do                                % For each class
    while (unassigned( $\lambda_t$ )  $\neq \emptyset$ ) do    % For each unassigned boundary point
         $b = \max(\text{weights}(\text{unassigned}(\lambda_t)))$ ;
         $[I] = \text{Organize\_GA\_Template}(t, b, \Gamma)$ ;
         $[\vec{z}] = \text{Greedy\_Search}(I)$ ;
         $\vec{x} = \text{ga}(t, b, \Lambda, I)$ ;                % Run the Genetic Algorithm
         $\lambda_t = \text{assign}(\lambda_t, \vec{x})$ 
    end; %while
end; %for t

```

Figure 55 Simplified Pseudo-code for the Region Identification Phase.

```

function [fitness] = objfun( $\vec{a}_i$ )                    % Evaluate the  $i$ th individual,  $\vec{a}_i$ 
 $\Lambda(:).valid = \text{true}$ ;                          % Assume all boundary points are members
for  $j = 1$  to  $l$  do                                  % For each enabled partition
    if ( $\vec{a}_i(j) > 0$ ) then
         $h = f_o(j) \times f_m(j)$ ;
        for  $k = 1$  to  $|\Lambda|$  do                    % For each boundary point
            if ( $\Lambda(k).valid$ ) then                % Is the boundary point still assigned?
                if ( $h(\Lambda(k)) < 0$ ) then          % Is the boundary point in the region?
                     $\Lambda(k).valid = \text{false}$ ;        % If not, eliminate from consideration
                end; %if
            end; %if
        end; %for k
    end; %if
end; %for j
return  $\Phi(t, \vec{a}_i, \Lambda)$ ;                        % Base fitness on remaining members

```

Figure 56 Simplified Pseudocode for the Objective Function.

As a starting point of reference for this discussion, Appendix A provides a detailed discussion of GAs, including a derivation of their complexity. Specifically, Gordon (46) derived the complexity of a binary GA as

$$O_{GA} = (q \times p \times \max(l, \Phi, n)) \quad (39)$$

where q is the number of generations, p is the size of the GA population, l is the length of the GA chromosome and Φ is the objective function. Thus, to determine the complexity of the GA, the complexity of the objective function must first be derived. In the GA employed by GRaCCE, the objective function evaluates the CH region defined by each individual in the population. As discussed in Section 4.5, each candidate region is represented using a binary chromosome, where each bit corresponds to a different partition. Because only those partitions associated with the target class are used in a given search, the length of the chromosome can vary between 1 and $|\Gamma|$; in general: $1 \leq l \leq (|\Gamma|/(m-1))$. For simplicity, however, it is assumed that the maximum workload is modeled by $l = |\Gamma|/m$ with all partitions enabled.

The objective function in the Figure 56 pseudocode computes fitness for one individual at a time. Starting with the first gene, all boundary points are tested to determine if they are members of the region defined by the chromosome. Once a boundary point fails for a given gene, it is removed from consideration when testing subsequent genes in the same chromosome. As a result, the pool of boundary point candidates should quickly converge to a small subset of Λ . Despite this, each point must still be checked for validity every time. Therefore, the number of operations required to evaluate a given chromosome (O_Φ) can be approximated by

$$O_{\Phi} = l \times n_{bp} \quad (40)$$

This initial estimate of O_{Φ} can be simplified in a number of ways. Both l and n_{bp} are related to the size of Γ . Since $|\Gamma|$ grows at the rate specified in Equation 38, each class can be expected to have $d + m$ associated partitions, so $l \approx d + m$. In turn, n_{bp} can be approximated as $m(d + m)$. This simplifies Equation 40 to

$$O_{\Phi} = m(d + m)^2 \quad (41)$$

Inserting this result into Equation 39 we obtain,

$$O_{GA} = (q \times p \times m(d + m)^2) \quad (42)$$

For a binary GA, Goldberg (41) recommends a population of size \sqrt{l} in order to ensure that all allele values are represented for each gene². Alternatively, research by Reeves (107) and Schaffer (112) show that small, constant populations suffice for binary string representations as string length increases. To be conservative, however, we utilize Goldberg's more pessimistic sizing estimate, which yields a population size of $\sqrt{d + m}$. Substituting this relation for p , the complexity estimate for the GA then becomes

²In his analysis, Goldberg assumes that only the crossover operator is used for reproduction - mutation is avoided. Of course, if mutation is used, then any schema is reachable.

$$O_{GA} = (q \times m(d + m)^{5/2}) \quad (43)$$

With Equation 43, we can compute the complexity of the RI phase. Since each class consists of a single mode, it is assumed that all boundary points for the target class are assigned in the first iteration of the main loop. Within the main loop of the pseudocode in Figure 55, the GA dominates the other operations. As a result, the complexity of the phase can be estimated as

$$O_{RIP} = (q \times m^2(d + m)^{5/2}), \text{ where } l \approx d + m \quad (44)$$

given a data set modeled by $\Omega(m, d)$. This result is interesting because it implies (for the average case) that time complexity is *independent* of data set size (at least for the RI phase). From an analytical standpoint, n_{bp} is maximized for a given data set when every training point is a boundary point (i.e., $m = n_{bp} = n$). While mining such a data set may prove impractical, it does result in a worst case complexity estimate. On the positive side, such an assumption makes other simplifications possible (such as $l \approx d$). Given this, the worst case time complexity for the RI phase is

$$O_{RIP} = (q \times n^2 d^{3/2}) \text{ where } l \approx d \quad (45)$$

While the complexity estimate in Equation 45 is polynomial with respect to data set size and dimensionality, it is important to remember that for the vast majority of data sets, the number of classes is much smaller than the number of

instances ($m \ll n$). It is also significant to note that the time complexity of the RI phase should decrease if the number of classes is less than the number of modes.

5.5.3 Comparison to Decision Tree Algorithms. With regard to complexity, it has been shown that decision tree induction (DTI) algorithms that use axis-parallel splits, such as ID3 and C4.5 (105), have a worst case growth rate of $O(nd^2)$ (144). In addition, Murthy's OC1 algorithm (94) (which uses oblique splits) has a worst case complexity of $O(dn^2 \log(n))$. While GRaCCE's worst case growth rate of $O(qn^2 d^{5/2})$ is much worse than C4.5, it is roughly competitive with OC1. Recall from Section 4.10, the RI phase can potentially achieve a linear (factor of m) improvement in time efficiency when executed in a distributed environment.

From a practical standpoint, however, the true complexity of each of these methods is driven more by the "internal" complexity of the data than it is by the raw size of the data set. Indeed, from an algorithmic perspective, it is intuitive that a more complex data set takes longer to process than a simpler one. Additionally, the data related assumptions outlined in Section 5.5.1 support the assertion that a data set's complexity can be *independent* of its size. This hypothesis is further bolstered by test results on the scalability of GRaCCE which are presented in Section 6.7.2. Given these factors, the worst case time complexity estimates may not be a realistic basis for comparing the efficiency of these algorithms.

Another factor that must be considered is the quality of the rule sets produced by each method. It has been demonstrated that GRaCCE generates partitions which reflect the natural structure of the data. In contrast, the DTI algorithms use "one size fits all" partitioning metrics which may not work as desired for a given data set. Considering the complexity discussion, a fundamental question to ask is: how effective are the rules produced by DTI algorithms? From a classification accuracy standpoint, the decision rules produced by DTI algorithms have "reasonable" accuracy while being inexpensive to compute (130). However, while desirable, these

characteristics do not guarantee that the rules are meaningful to the end user. As discussed in Section 3.7, over-fitting can result in very accurate rules that unnecessarily fragment the data space; this can lead to a situation where the forest cannot be seen for the trees. Since data mining applications often put a premium on *meaningful* decision rules, these must reflect the natural structure of the data. Given this, the superior efficiency of DTI algorithms may be a misleading indicator of their effectiveness in the data mining domain.

5.5.4 Derivation of Concurrent Growth Rate. In Section 4.10, an architecture for a concurrent version of GRaCCE (cGRaCCE) is proposed. The distinguishing characteristic of the architecture is that it executes the CH searches for each class in parallel. This approach is feasible because all the necessary inputs to each search (such as the set of boundary points and partitions) are generated prior to the start of the phase. In addition, no dependencies exist between searches for each of the m classes. This is because each CH region search only impacts resources related to its target class (such as boundary point assignments). As a result, the region identification phase for each class can be treated as a series of independent tasks, each executed on its own processor. Thus, the execution time of the RI phase can *potentially* be reduced by a factor of m (given m processors), yielding a growth rate of

$$O_{cRIP} = (q \times m(d + m)^{3/2}) \quad (46)$$

In the next chapter, the run-time execution speedup resulting from this architecture is measured for a number of data sets. It is important to note that several studies (1; 132; 154) have demonstrated that distributed DTI algorithms achieve *near linear* speedup as the number of CPUs is increased due to the use of techniques to balance processing load among the processors.

5.6 Summary

This chapter developed the theoretical foundation for the GRaCCE data mining algorithm. While a broad range of topics were discussed, the unifying theme is that, unlike other rule induction algorithms, this system evolves rules based on the natural structure of the data. This assertion is supported by theorems which prove the GRaCCE algorithm has the following properties:

- It winnows the training data (by applying the edited kNN procedure) to a subset whose boundary point pairs straddle the Bayes decision surface.
- GRaCCE can utilize the neighborhood around each boundary point (in a given pair) to generate a complete set of partitions which approximate the Bayes decision surface in a piecewise fashion.
- Selected partitions can be combined to locate CH regions within the data. It is further shown that GRaCCE is capable of finding optimal CH regions as defined by a objective function given sufficient time.

In addition, the fitness landscape for the RI phase was analyzed for a representative data set. The purpose here was to provide some additional justification for our use of GAs to perform a hard local search for a CH region, given an initial solution. The last part of the chapter focused on deriving the time complexity of GRaCCE's RI phase. The motivation behind this exercise is to show that the execution efficiency of the CH region search is primarily influenced by characteristics which reflect the structural complexity of the data (such as classes/modes and dimensionality). These results show that the average case time complexity for the RI phase is independent of the size of the data set. Although a worst case complexity was also derived, the fact that this reflects an unrealistic case makes it difficult to compare the system's efficiency to those of decision tree algorithms.

VI. Experiments with GRaCCE

The preceding chapters explained the algorithmic and theoretical underpinnings behind GRaCCE. The value of any system, however, ultimately depends on how it performs in practice; accordingly, the focus of this chapter is testing. The system's relative utility is evaluated by comparing its performance to that of several decision tree algorithms on a suite of benchmark data sets. The metrics of interest are the accuracy and complexity of the induced decision rule set. In addition, the system's robustness is tested with respect to its user-tunable parameters. Lastly, a concurrent version of GRaCCE is demonstrated.

6.1 Objectives

When evaluating any system, the test objectives should ensure that the system satisfies its original design goals. With this in mind, Table 14 provides a mapping between GRaCCE's design goals and the objectives of the tests needed to verify them. These objectives are the basis for the tests described later in the chapter. The one glaring omission from this list of objectives is a direct comparison of the run time execution performance of GRaCCE to the other rule induction algorithms. This testing is not accomplished here because GRaCCE is written in *MatLab*. Since *MatLab* code executes much slower than compiled *C++* code, such a test would only serve to verify the obvious. Before a meaningful comparison can be done, the software must be completely translated to *C++*.

6.2 Selection of Data Sets

In order to properly validate GRaCCE's credentials as a general purpose data mining algorithm, it is essential to determine how it performs on a wide variety of data sets. To attain this level of diversity, the data set selected for testing must vary in terms of the following characteristics:

Table 14 Mapping of Design Goal to Test Objectives.

Design Goal	Test Objective
Be a general purpose rule induction method.	Show that GRaCCE successfully processes a variety of different data sets.
Generate decision rule sets that are compact and simple.	Measure size and complexity of the generated rule set relative to competing methods.
Achieve comparable classification accuracy with other rule induction methods.	Compare the classification accuracy of the generated rule set relative to competing methods.
Be robust with respect to user defined parameters.	Demonstrate that GRaCCE's performance does not significantly change in response to incremental changes to user specified parameters.
The GRaCCE architecture can be parallelized.	Demonstrate a concurrency version of the system. Evaluate its run-time execution performance in both single and multi-processor configurations as data set size and complexity are varied.

- Number of classes
- Dimensionality
- Number of modes per class (i.e, unimodal versus multi-modal data)
- Size of data set (in terms of number of instances)
- Attribute type (discrete, continuous, mixed)

It is also important to evaluate GRaCCE using both synthetic *and* real data. The synthetic data discussed earlier served mainly to exercise the system's capabilities for specific scenarios. In contrast to the clean-room like perfection of synthetic data, "real-world" data sets often contain significant levels of noise, anomalies, and missing data. Consequently, testing on such data gives a much clearer picture of a system's expected performance.

With these considerations in mind, the ten real-world data sets selected for testing are described Table 15. With the exception of FLIR, each of these data sets

Table 15 Real Wold Data Set Descriptions.

Name	Classes	Samples (Original)	Features	Types of Features
Iris	3	150	4	Continuous
Cancer	2	699	9	Continuous
Wine	3	178	13	Continuous
Glass	6	214	9	Continuous
Diabetes	2	768	8	Continuous
FLIR	2	1000	6	Continuous
Mushroom	2	8124	22	Discrete
Thyroid	2	3163	25	Mixed
Ionosphere	2	351	34	Continuous
Soybean	19	307	35	Discrete

are frequently cited benchmarks available from the University of California (UC) - Irvine Machine Learning Repository. The FLIR data was generated by Ernisse (30) as part of a project to develop an image based Automatic Target Recognition (ATR) system for SCUD missile launchers. These data sets augment the synthetic ones introduced in Chapter III. While it is always possible to evaluate the system with additional data, the sets selected are diverse with respect to the previously outlined characteristics. Other data sets from the Irvine repository (such as Voter and Credit) were considered and rejected because they contributed little to the breadth of the test suite.

6.3 Data Preparation

Proper preparation of the target data set is a critical part of any data mining operation. For these experiments, preparation consists of feature selection and replacement of missing data. With regard to the former activity, recall that reduction of a data set's dimensionality is the best way to fight the *curse of dimensionality* (8). To accomplish this, each of the data sets was run through the GRaCCE feature selection procedure (described in Section 4.2.1). This process returned those features which proved to be the best class discriminators for the kNN algorithm. Table 16

Table 16 Reduced Data Set Descriptions.

Name	Features (Reduced)	Selections
Iris	2	3, 4
Cancer	6	1, 5, 6, 7, 8, 9
Wine	3	7, 9, 10
Glass	4	3, 4, 6, 8
Diabetes	5	2, 4, 6, 7, 8
FLIR	2	4, 6
Mushroom	4	5, 20, 21, 22
Thyroid	2	15, 23
Ionosphere	14	2, 4, 7, 12, 13, 15, 16, 18, 19, 25, 26, 32, 34
Soybean	15	2, 3, 4, 5, 8, 14, 15, 20, 22, 27, 28, 29, 31, 32, 35

contains the reduced feature list for each data set. Note that this procedure was not performed on the synthetic data sets since they already have a minimal function set.

Of those listed in Table 15, only the Mushroom, Soybean and Thyroid data sets had instances with missing attributes. While several techniques for repairing missing data were discussed in Section 3.8, the most sophisticated of these were rejected due to their extreme complexity and unsuitability for data mining applications. Instead, the data sets were repaired using a kNN based imputation method. This method fills in each missing attribute based on the k closest instances (of the same class) which possess that attribute. To compensate for instances missing multiple features, the total distance between two feature vectors is computed as the average of the difference between all attributes which are not missing. While this approach could produce poor results for sparse data, the data sets utilized here did not have this characteristic.

Once the k closest instances are selected, the missing value is computed. For real-valued attributes, the missing value is assigned the average of that attribute for the k nearest instances. For discrete attributes, the missing value is assigned to the majority value of the k nearest instances. Although filling in each missing attribute

with a fixed value introduces some bias, this step is necessary in order to generate a single, unambiguous set of decision rules. In addition, the level of bias is equal for each algorithm tested. As a result, this approach is acceptable for purposes of algorithm comparison.

6.4 Reporting of Results

Since the backbone of any test regimen is the supporting documentation, this section explains how the results for each run are recorded. During the course of each GRaCCE run, the results are written to a report file. This report contains the following information:

- Data file information, including class specific tallies for the training and test sets.
- Algorithm parameter settings.
- Feature selection status.
- Error rate tallies (for training, evaluation, and test sets).
- Description of final partition set, including the average decrease in partition dimensionality resulting from the partition simplification phase.
- Description of CH regions (clusters).
- A mapping of CH regions to the partitions that define them.

An example report file is located in Appendix C. Because the GRaCCE algorithm has distinct phases, the classification error rate is reported at the end of each phase where a viable rule set exists; this corresponds to the last three phases. Every report file contains several labeled error rates, each corresponding to the part of the data set (train, eval or test) being evaluated and the particular test method. An explanation of these terms is provided in Tables 17 and 18 for the data set and method categories, respectively.

Table 17 Classification Error Rate Descriptions.

Category	Classification Error Computed On:
Train	Portion of training data that remains after winnowing is complete.
Eval	Entire set of training data.
Test	Entire set of test data.

Table 18 Test Method Descriptions.

Test Method	Description
Method 1	Based on partitions derived from Region Identification Phase.
Method 2	Based on partition set resulting from the Region Refinement Phase.
Method 3	Reflects the use of Mahalanobis distance to augment either partitions set used in Method 1 or Method 2. The selected partition set is the one that results in the lowest error rate.
Method 4	Based on partition set resulting from the Partition Simplification Phase.
Method 5	Reflects the use of Mahalanobis distance to augment the partition set used in Method 4.

6.5 Inter-Algorithm Comparison

The purpose of this section is to evaluate the performance of GRaCCE against that of other rule induction algorithms. Such a comparison provides the reader with a perspective of how the system's performance relates to techniques which represent the current state-of-practice in this domain.

6.5.1 Test Methodology. With regard to the test objectives stated in Table 14, the first three are assigned to this section. In order to gauge the relative utility of GRaCCE in terms of these objectives, performance is measured with respect to the following criteria:

- Classification Accuracy
- Generalization
- Decision Rule Set Complexity

The degree of classification accuracy for each algorithm (reported in terms of the error rate) is determined by the accuracy of its generated rule set. Generalization is the absolute difference between the training and test set error rates. The complexity of the decision rule set generated by each algorithm is characterized by four separate metrics: the average number of rules, conditions per rule, terms per rule set and compactness¹.

Each test case is performed on two versions of each data set, containing the full and reduced feature sets (see Table 16). The results are tabulated on the basis of four, five fold cross-validations (for a total of twenty runs per data set version). Prior to each cross-validation sequence, the data set being tested is randomly shuffled. The data is then divided up into five equal sized folds. For each run, four folds compose the training set and the fifth is used for testing. Over the course of the cross-validation sequence, each fold is rotated into the test set. All classification accuracy results are based on performance against the test data set.

6.5.2 GRaCCE Specific Considerations. As discussed in Section 6.4, each GRaCCE report includes the error rates specified in Table 18. Given this, it is essential to establish guidelines for using the information in the report file to tabulate results that support the test objectives discussed above. Since the comparisons performed in this section are between generated *rule sets*, the metric computations exclude Methods 3 and 5, which go beyond the information provided in each rule set. For the classification accuracy and generalization computations, the lowest of the error rates given by Methods 1, 2 and 4 are chosen. Depending on which error rate is chosen, there can be a substantial difference in rule set complexity since the least complex rule set does not necessarily have the lowest error rate. The rule set size and complexity metrics are derived, in turn, from the rule set of the test method used in the classification accuracy computation. These decisions are made

¹A more in-depth description of these metrics is provided in Section 6.5.7

independently on a *per run* basis. Thus, the results for GRaCCE reported in this section can be characterized as the composite for Methods 1, 2 and 4 with error rate determining the selected method.

6.5.3 Choices for System Comparison. In our evaluation, GRaCCE's performance is compared to that of several decision tree algorithms. As discussed earlier, decision tree induction has proven to be a quick and reliable method of generating high quality classification rules. Since both types of algorithms produce rules as output, a comparison between them is far more meaningful than if GRaCCE were evaluated against a black box type of classifier (such as a neural network). In addition, the decision tree software packages were readily available, both at AFIT and on the Internet.

The particular systems chosen were C4.5, CART and OC1; a comprehensive description of these algorithms was presented in Chapter II. Because each of these algorithms uses a different approach to decision tree induction, testing against all three provides a balanced picture of GRaCCE's relative performance. The C4.5 algorithm searches for partitions that are orthogonal to the feature axis. Because of this constraint, C4.5 tends to produce rule sets that are large, but easily understood. In contrast, CART searches for oblique partitions composed of linear combinations of features. As a result, we expect CART to produce smaller rule sets with more complex antecedents than C4.5. The version of CART used in these tests is implemented as part of the LNKnet pattern recognition test-bed (82). Lastly, the OC1 algorithm (like CART) also searches for oblique partitions. The key difference between the two, however, is that OC1 uses limited random search while the CART method is entirely deterministic.

6.5.4 Algorithm Parameters. This section describes the parameters used in by each algorithm in the experiments. In some cases, the same parameters (nominally

Table 19 GA Parameter Settings.

Parameter Name	Value
Population Size	100
Probability - Crossover	70.0%
Probability - Mutation	10.0%
Probability - Replacement	90.0%
Convergence Window Size (q)	10 generations
Recombination Function	2 Point Crossover
Selection Function	Stochastic Universal Sampling (SUS)

the defaults) were employed for all experiments. In other cases, it was necessary to modify the parameters for each data set in order to achieve optimal performance.

6.5.4.1 GRaCCE Specific Parameters. The parameters used by the GRaCCE algorithm can be grouped into two basic categories: Genetic Algorithm and Region Identification (RI) phase. The GA parameter set is used to initialize and control the genetic search of each CH region. In these experiments, these parameters (summarized in Table 19) were held constant for all test cases. For reference, Appendix A contains a comprehensive explanation of how these parameters affect the GA. Table 20 lists the settings (by data set) for the most critical parameters from the RI phase. The specific role played by each of these parameters are discussed in Chapter IV and Appendix B.

6.5.4.2 C4.5 Parameters. When running C4.5, the default parameter settings were utilized. The parameter of most importance is the confidence level used for pruning the decision tree; this parameter is set to 25%. It is also important to note that the windowing option was not enabled; as a result, only a single tree was constructed for each run.

6.5.4.3 CART Parameters. The version of CART utilized for these experiments offers few user-selection settings. Of those available, however, both axis-parallel and oblique types of splits are enabled for consideration. In addition,

Table 20 GRaCEE - Region Identification Phase Parameter Settings.

Data Set	Parameter				
	Partition Set Type	δ	γ_t	α	β
Diabetes	Global	0.80	1.0	0.5	0.2
Wine	Mixed	0.80	1.0	0.5	0.2
Cancer	Mixed	0.80	1.0	0.5	0.2
FLIR	Mixed	0.95	0.8	0.5	0.2
Iris	Mixed	0.80	1.0	0.5	0.2
Glass	Mixed	0.80	1.0	0.5	0.2
Mushroom	Mixed	0.80	1.0	0.5	0.2
Thyroid	Mixed	0.80	1.0	0.5	0.2
Ionosphere	Mixed	0.90	1.0	0.5	0.2
Soybean	Mixed	0.80	1.0	0.5	0.2

Table 21 CART Parameter Settings.

Data Set	Stop if $\leq N$ patterns
Diabetes	25
Wine	25
Cancer	50
FLIR	50
Iris	25
Glass	10
Mushroom	25
Thyroid	50
Ionosphere	10
Soybean	5

expansion of the tree at a given node is halted if there is zero node error or if there are less than N patterns assigned to the current node. During testing, it was found that exercising this option resulted in smaller trees with better classification accuracy. The settings utilized for N , which produced the *best* results for each data set (in terms of classification accuracy) are shown in Table 21.

6.5.4.4 OC1 Parameters. The parameters defined within the OC1 algorithm are those that control the random search at each node for better hyperplanes. In particular, the parameters and corresponding default settings are summa-

Table 22 OC1 Parameter Settings.

Parameter Description	Value
Number of restarts for initial hyper-plane selection	20
Order of coefficient perturbation	Sequential
Number of random perturbations tried at each local minimum	5

rized in Table 22. Note that our experiments utilized only these default settings. A more detailed explanation of how these parameters control algorithm execution can be found in Section 2.2.5.1.

6.5.5 Classification Accuracy. The error rates of the decision rule sets generated by each method for the full and reduced feature sets are summarized in Figures 57 and 58, respectively². These results show that out of 40 comparisons, GRaCCE was significantly more accurate 11 times and was outperformed 8 times for the full feature set; the corresponding numbers for the reduced feature set experiments were 10 and 8, respectively. While the system performed slightly better than its competition for both types of feature sets, it does not appear to consistently outperform any of the methods surveyed. It is important to note, however, that GRaCCE did worst on those data sets containing discrete features (refer to Table 15); this trend is especially evident when the algorithm is compared to C4.5 and OC1. This finding indicates that the system may have to be modified to better accommodate discrete data sets.

6.5.6 Generalization. In Section 3.7 the problem of classifier over-fitting was discussed. Recall that over-fitting occurs when the decision rule set reflects the training examples, but little else. This condition frequently results in excellent accuracy on the training data set and poor accuracy on the test set. It can therefore

²The superscript codes indicate which methods a given result is statistically superior to. This determination is made on the basis of a difference of means test (129) at a 0.05 level of significance. The code for each method is as follows: GRaCCE (1), CART (2), C4.5 (3), and OC1(4). These codes are also used for Figures 67 and 68.

Data Set	Error Rate ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	$22.8 \pm 2.5^{2,3,4}$	27.7 ± 4.0	28.1 ± 2.5	27.6 ± 3.0
Wine	6.0 ± 3.4^4	6.6 ± 3.6	6.5 ± 4.8	11.3 ± 6.1
Cancer	$4.2 \pm 1.7^{2,3}$	5.8 ± 1.9	5.3 ± 1.5	4.9 ± 1.4
FLIR	25.6 ± 2.7	24.1 ± 2.1^1	24.3 ± 2.8	23.5 ± 2.7^1
Iris	$2.9 \pm 3.5^{2,3}$	5.5 ± 3.6	5.3 ± 3.2	3.7 ± 3.0
Glass	36.4 ± 6.2	35.7 ± 5.7	33.9 ± 6.8	37.0 ± 5.8
Mushroom	8.5 ± 3.3^2	14.8 ± 10.4	0.0 ± 0.0^1	0.6 ± 0.8^1
Thyroid	2.4 ± 0.9	1.4 ± 0.7^1	1.0 ± 0.3^1	1.7 ± 0.5^1
Ionosphere	10.5 ± 2.9	12.3 ± 3.0	10.1 ± 3.3	13.2 ± 5.2
Soybean	$10.7 \pm 2.8^{2,4}$	40.4 ± 15.6	6.7 ± 1.8^1	13.5 ± 4.1

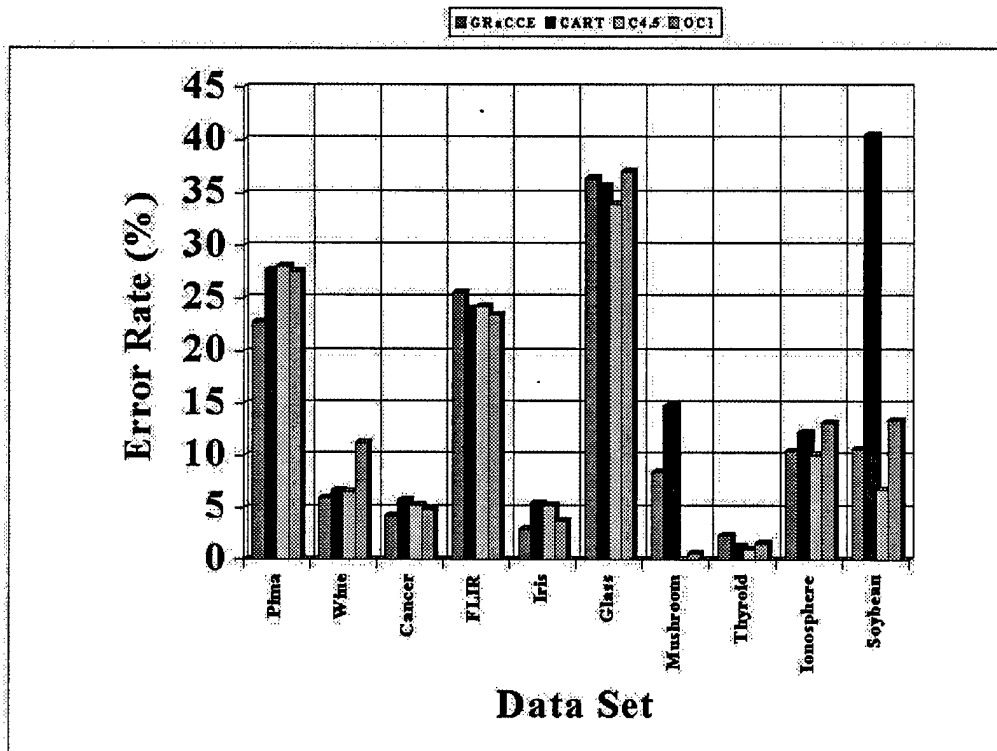


Figure 57 Baseline Comparison - Classification Accuracy for Full Feature Set (Rules Only).

Data Set	Error Rate ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	$22.8 \pm 2.2^{2,3,4}$	25.9 ± 3.6	28.3 ± 3.4	27.6 ± 3.8
Wine	$7.1 \pm 3.4^{2,4}$	12.1 ± 5.3	8.8 ± 4.2	15.7 ± 12.4
Cancer	4.5 ± 2.0	4.9 ± 1.7	5.4 ± 2.0	5.1 ± 1.9
FLIR	$23.6 \pm 3.5^{3,4}$	25.3 ± 3.0	26.5 ± 1.4	25.8 ± 2.4
Iris	3.0 ± 2.6	4.5 ± 3.6	4.3 ± 2.5	7.0 ± 9.3
Glass	34.1 ± 4.7^2	38.4 ± 4.6	32.3 ± 5.1	34.6 ± 8.7
Mushroom	2.8 ± 2.1^2	6.3 ± 2.2	0.0 ± 0.0^1	0.1 ± 0.1^1
Thyroid	2.3 ± 1.0	0.8 ± 0.2^1	1.1 ± 0.4^1	1.2 ± 0.6^1
Ionosphere	13.0 ± 6.1	11.0 ± 3.7	9.1 ± 3.0^1	11.1 ± 3.4
Soybean	15.7 ± 5.3^2	43.7 ± 13.2	9.4 ± 2.6^1	12.1 ± 3.4^1

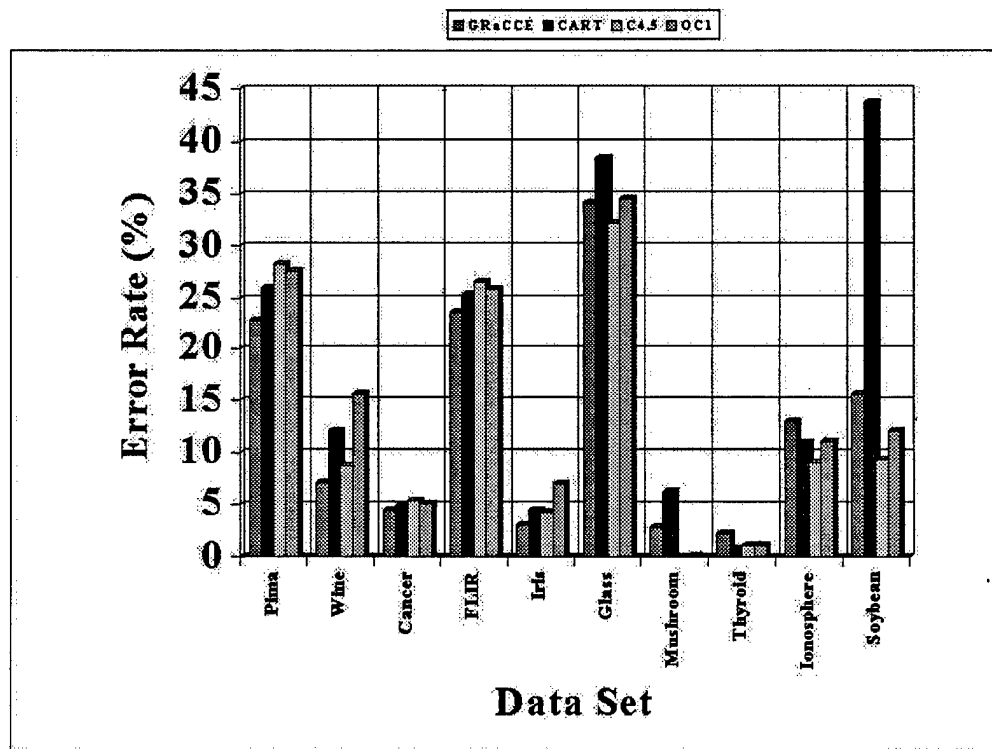


Figure 58 Baseline Comparison - Classification Accuracy for Reduced Feature Set (Rules Only).

be concluded that the degree of over-fitting is inversely proportional to how well an algorithm generalizes. Generalization, in turn, can be measured by taking the mean of the absolute difference in error rate between the training and test set; the smaller the difference, the better the algorithm generalizes.

Over-fitting may appear attractive to anyone whose goal is to optimize classification accuracy, because it achieves a very low training error rate. Since training data is 80% of the total, the *overall* error rate may be lower for those techniques which generalize poorly relative to those that generalize well. While this is a seductive argument, it has two major flaws. The first is that accuracy is almost universally judged based on the test case results. Second, solutions that over-fit the data tend to be more complex as compared to those with better generalization (51). This rule of thumb is consistent with the principle of *Occam's razor* (10), which tells us to always choose the simplest model that fits the data.

Accordingly, a comparison of the generalization performance for the GRaCCE, CART and C4.5 algorithms is shown in Figures 59 and 60³. These results show that GRaCCE generalizes better than CART for all data sets and better than C4.5 for all but two data sets (Mushroom and Thyroid). Given the above discussion, we should expect the GRaCCE generated decision rule set to be less complex than those of CART and C4.5. The actual findings on this point are discussed in the next section.

6.5.7 Decision Rule Set Complexity. In this section, we quantitatively analyze the complexity of the rule sets that yield the previously reported results. From a data mining standpoint, complexity metrics are important because they relate the ease with which the rule set can be interpreted; in short, the simpler the rule set, the easier it should be to understand. In addition, simpler rule sets tend to be more meaningful than complex ones for equivalent levels of accuracy. This is because individual rules have greater coverage of the data when they are fewer in

³Note that results for OC1 are not included because its reports do not contain training data error.

Data Set	Δ Error Rate ($\mu \pm \sigma$)		
	Method		
	GRaCCE	CART	C4.5
Diabetes	2.7 ± 1.7	27.3 ± 4.1	21.1 ± 3.7
Wine	3.6 ± 2.9	6.5 ± 3.5	5.5 ± 4.5
Cancer	1.7 ± 1.5	5.3 ± 1.9	3.3 ± 1.8
FLIR	2.4 ± 1.9	23.4 ± 2.1	17.3 ± 2.9
Iris	2.7 ± 2.6	5.4 ± 3.6	3.8 ± 3.0
Glass	6.9 ± 3.3	35.3 ± 5.7	26.7 ± 7.1
Mushroom	0.6 ± 0.5	6.7 ± 3.8	0.0 ± 0.0
Thyroid	0.8 ± 0.8	1.4 ± 0.7	0.4 ± 0.4
Ionosphere	2.8 ± 2.3	11.8 ± 3.0	8.1 ± 3.6
Soybean	2.7 ± 2.3	25.3 ± 8.6	3.8 ± 1.8

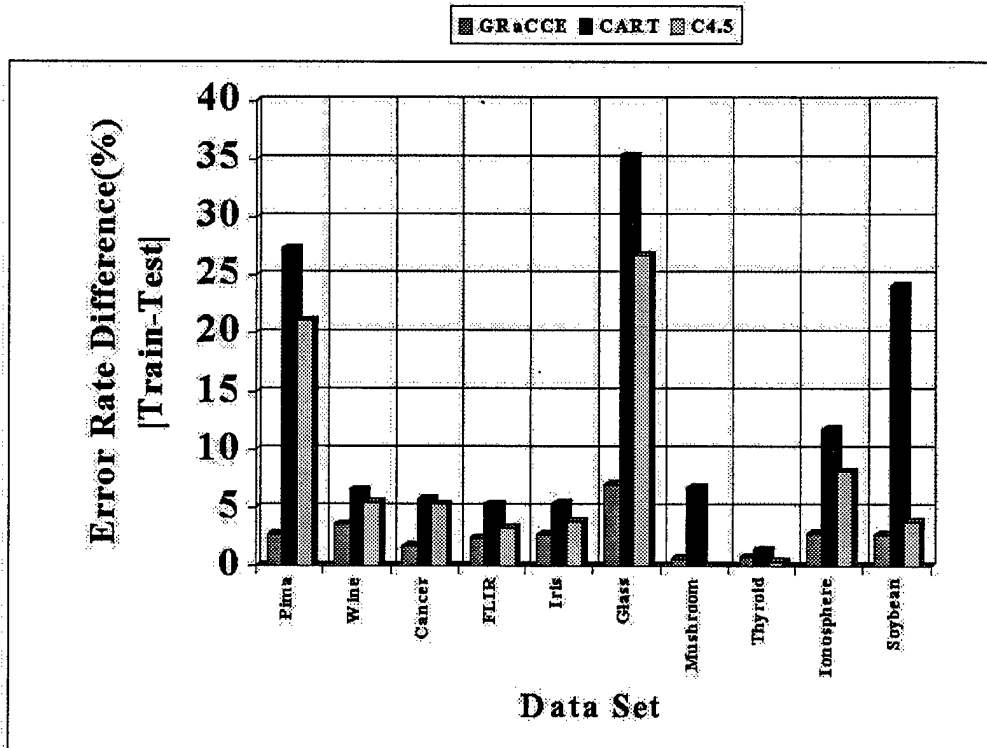


Figure 59 Baseline Comparison - Generalization Performance on Original Feature Set.

Data Set	Δ Error Rate ($\mu \pm \sigma$)		
	Method		
	GRaCCE	CART	C4.5
Diabetes	2.1 ± 1.5	24.4 ± 4.1	19.5 ± 3.6
Wine	3.4 ± 2.5	10.6 ± 5.5	5.9 ± 4.3
Cancer	1.9 ± 1.3	4.3 ± 1.7	2.9 ± 2.1
FLIR	3.6 ± 2.1	24.5 ± 2.9	12.9 ± 2.1
Iris	2.6 ± 1.9	4.5 ± 3.5	2.9 ± 2.0
Glass	5.3 ± 3.2	34.9 ± 5.0	16.7 ± 5.5
Mushroom	0.3 ± 0.3	3.7 ± 1.0	0.0 ± 0.0
Thyroid	0.5 ± 0.4	0.8 ± 0.2	0.3 ± 0.4
Ionosphere	3.8 ± 3.0	11.0 ± 3.8	6.8 ± 3.0
Soybean	3.3 ± 2.6	27.9 ± 5.6	3.3 ± 2.3

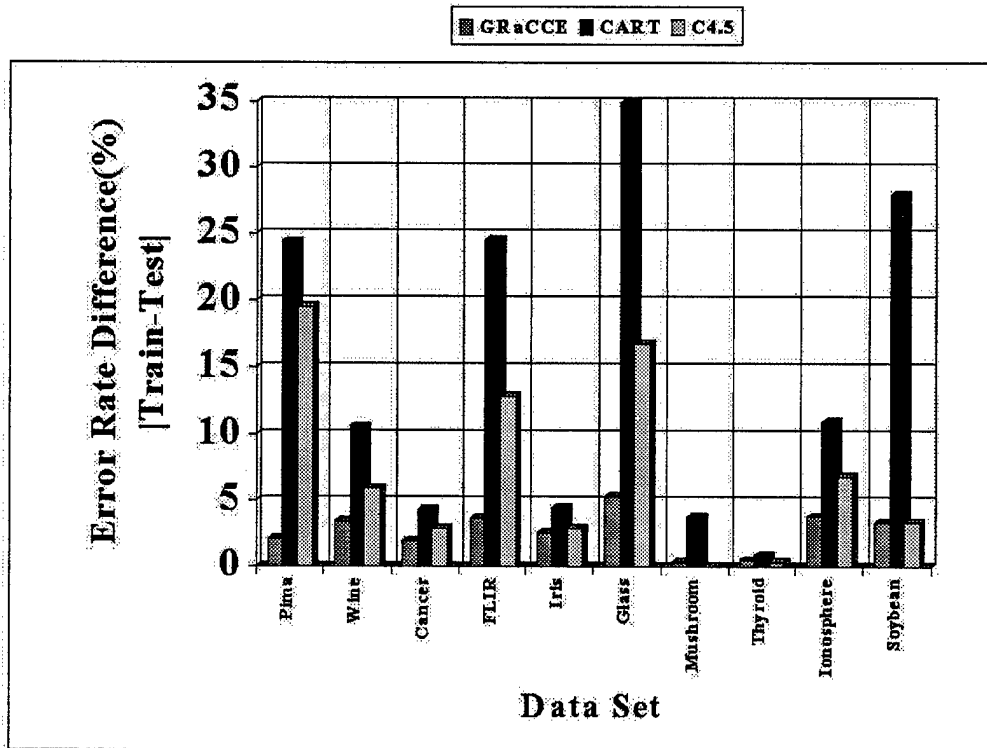


Figure 60 Baseline Comparison - Generalization Performance on Reduced Feature Set.

Table 23 Description of Rule Set Metrics.

Metric (μ)	Description
Number of Rules	For GRaCCE, this equates to the number of CH regions. For the decision tree algorithms, this corresponds to the number of leaves on the tree.
Conditions per Rule	For GRaCCE, this metric represents the number of partitions per CH region.
Compactness	This metric provides an estimate of complexity that is independent of partition type (i.e., univariate or oblique). It is computed by multiplying the number of rules in each decision rule set by the average number of conditions per rule.
Terms per Rule Set	This is the sum of the number of terms used in each condition over all rules in a given set. Algorithms which use oblique partitions (i.e., GRaCCE, CART, and OC1) can have up to d terms per condition. For those using univariate partition (i.e., C4.5), each condition has a single term. Thus, unlike compactness, this complexity metric penalizes algorithms for using oblique partitions.

number. To this end, the four metrics collected for each data set are described in Table 23; each measures a different, yet important, aspect of rule set size.

The corresponding measurements of each metric (for both the original and reduced feature sets) are shown in Figures 61 through 68. If these results are viewed in terms of compactness (refer to Figures 67 and 68), then GRaCCE clearly yields less complex decision rule sets; for almost every case, it generated rule sets that were much smaller than those of its competition. In several instances, GRaCCE found solutions that had less than half of the rules output by C4.5 or CART. These results are consistent with our earlier published findings (89). It must be noted, however, that while the rule sets produced by GRaCCE were statistically more compact than those of OC1, in absolute terms the difference was often marginal. As indicated in Table 20, GRaCCE typically used a mixed set of partitions (i.e., both global and local) to isolate CH regions. The notable exception to this is the Diabetes data set; in this case, the global partitions provided the superior solution (a rare phenomenon).

The difference in rule set size can be attributed to the way GRaCCE generates and evaluates decision boundaries. Recall from Chapter V that GRaCCE partitions approximate the Bayes optimal (natural) decision boundaries. In addition, the algorithm searches for the best CH regions (in terms of accuracy and coverage) using combinations. As a result, GRaCCE can form CH regions that better reflect the natural clustering structure of the data. Contrast this with decision trees, which tend to partition data based on statistical metrics computed for a given data subset. While effective, these splitting criteria often ignore the natural boundaries within the data. Also, because partitions are added one at a time, the synergistic effect of partition combinations is ignored. This often results in the construction of rule sets that are more complex than necessary.

For many data sets, however, the natural class boundaries can only be approximated using oblique partitions (each of which can have up to d terms). As Figures 65 and 66 show, using these types of partitions adds more terms to each rule set; this makes it harder to interpret each condition. When viewed from this perspective, the complexity results are decidedly more mixed. For the original feature set, C4.5 outperforms GRaCCE for half the data sets; this figure falls to 20% for the reduced feature set, however. Even so, GRaCCE's performance is impressive considering the built in advantage C4.5 enjoys in this area (its conditions only have a single term). It is important to keep in mind, however, that C4.5 also generates the most decision rules per data set. As a result, the complexity is merely shifted from one metric to another.

Another apparent trend is the large variance in the compactness metric for the decision tree algorithms. Specifically, it appears that the complexity of the rule set is heavily dependent on the fold of the data withheld from training during the cross-validation process. These findings indicate that these techniques are fragile in terms of their ability to generate compact rule sets. In contrast, the GRaCCE generated results are more consistent across runs. This phenomenon is especially apparent in

data sets such as FLIR, Glass and Soybean. As discussed in Section 3.7, similar results have been reported by other researchers (108). For instance, Kohavi (75) suggests using up to 90% of data for training decision trees to achieve more consistent results. This is a major shortcoming because very large data sets are typically processed by sampling a small fraction of the available instances; mining is then performed on this subset (125). Given this, it is desirable to employ techniques which deliver consistent results when provided with an arbitrarily random data subset.

6.5.8 Viewing the "Big Picture". Thus far the algorithm comparison has been conducted by reviewing each metric in isolation. To characterize overall performance, however, it is essential to analyze metrics in combination. This goal is met by Figures 69 and 70, which compare the normalized accuracy and compactness metrics for each data set across all algorithms. These figures show a significant number of GRaCCE symbols clustered in the lower left hand corner of each graph; this indicates that the system consistently induces compact rule sets with better than average accuracy. Of its competitors, only the OC1 algorithm's performance is comparable to GRaCCE. These results indicate that the decision rule sets produced by GRaCCE tend to have better coverage and fewer conditions while achieving comparable accuracy with the decision tree algorithms. Because the rules found by GRaCCE have better coverage, they can be considered more meaningful than those of C4.5 (even if they are somewhat harder to interpret).

6.6 Intra-Algorithm Comparison

In this section, the various modes of the GRaCCE algorithm are compared; emphasis is placed on the relative tradeoff in performance associated with each.

6.6.1 Effect of Classifying Orphan Data. When the generated rule set is the sole basis for classification, there is no guarantee a given feature vector will be assigned to any class. This occurs because the CH regions found by GRaCCE do

Data Set	Number of Rules ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	2.0 ± 0.0	27.9 ± 1.9	55.2 ± 8.0	7.7 ± 6.7
Wine	3.3 ± 0.6	4.9 ± 0.9	5.3 ± 0.9	4.3 ± 1.9
Cancer	2.0 ± 0.0	5.4 ± 0.9	10.8 ± 3.0	2.6 ± 1.0
FLIR	7.8 ± 5.4	22.4 ± 1.6	65.3 ± 10.2	15.8 ± 24.1
Iris	3.0 ± 0.0	4.4 ± 0.5	4.3 ± 0.5	3.1 ± 0.2
Glass	7.4 ± 0.8	18.6 ± 1.4	23.6 ± 7.6	10.1 ± 6.4
Mushroom	8.3 ± 1.9	6.1 ± 3.7	15.4 ± 1.8	8.4 ± 2.4
Thyroid	2.8 ± 1.0	11.2 ± 2.2	7.0 ± 3.2	5.6 ± 4.5
Ionosphere	2.3 ± 0.6	8.3 ± 2.9	10.8 ± 2.4	3.6 ± 2.3
Soybean	20.8 ± 0.9	22.8 ± 11.6	31.6 ± 3.0	24.3 ± 5.9

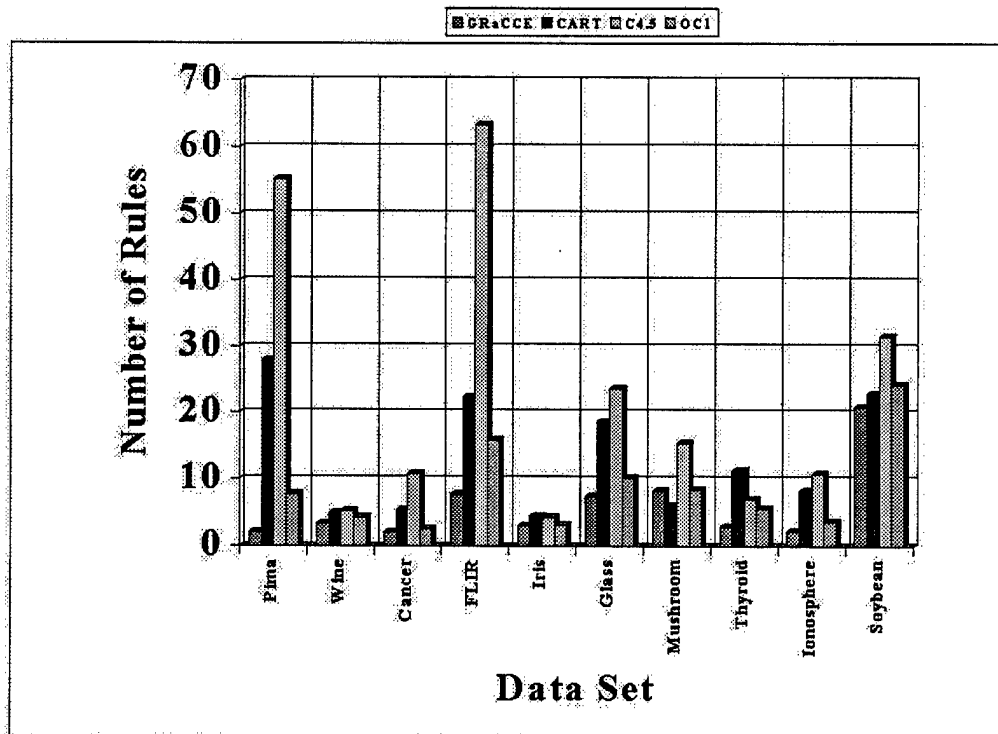


Figure 61 Baseline Comparison - Average Rule Set Size for the Original Feature Set.

Data Set	Number of Rules ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	2.0 \pm 0.0	17.1 \pm 1.1	53.8 \pm 7.7	5.4 \pm 7.4
Wine	4.3 \pm 0.6	4.5 \pm 0.5	6.3 \pm 1.4	3.3 \pm 1.6
Cancer	2.6 \pm 0.7	5.4 \pm 0.8	9.9 \pm 2.1	4.7 \pm 5.0
FLIR	7.7 \pm 3.2	23.0 \pm 2.1	49.8 \pm 11.1	18.0 \pm 24.7
Iris	3.0 \pm 0.0	4.9 \pm 0.4	4.6 \pm 0.9	3.4 \pm 1.6
Glass	5.9 \pm 1.0	9.1 \pm 0.9	17.7 \pm 3.5	10.0 \pm 4.9
Mushroom	6.0 \pm 0.9	7.2 \pm 2.4	33.0 \pm 0.0	11.4 \pm 1.0
Thyroid	3.4 \pm 2.0	10.7 \pm 1.5	6.0 \pm 3.2	6.5 \pm 3.1
Ionosphere	5.0 \pm 2.6	8.3 \pm 1.4	11.6 \pm 3.3	4.9 \pm 3.3
Soybean	22.9 \pm 3.9	16.2 \pm 7.7	35.1 \pm 2.2	20.4 \pm 2.7

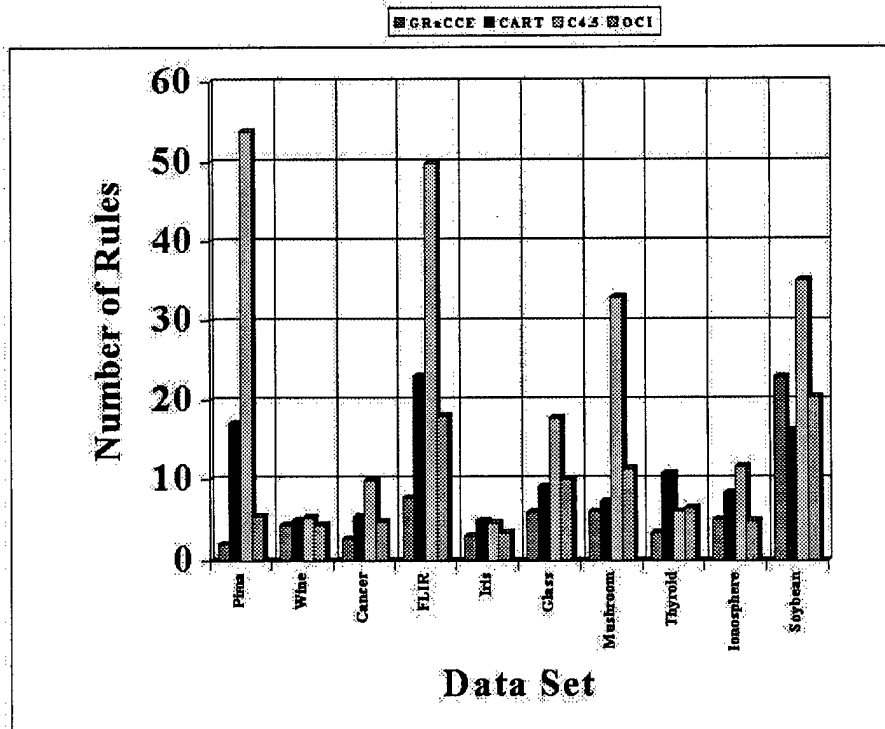


Figure 62 Baseline Comparison - Average Rule Set Size for the Reduced Feature Set.

Data Set	Conditions per Rule ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	1.0 ± 0.0	4.8 ± 0.1	5.8 ± 0.2	2.5 ± 1.1
Wine	1.6 ± 0.3	2.3 ± 0.3	2.4 ± 0.2	2.0 ± 0.6
Cancer	1.3 ± 0.4	2.4 ± 0.2	3.4 ± 0.4	1.3 ± 0.4
FLIR	2.0 ± 0.8	4.5 ± 0.1	6.0 ± 0.2	2.9 ± 1.7
Iris	1.4 ± 0.1	2.1 ± 0.2	2.1 ± 0.2	1.6 ± 0.1
Glass	2.0 ± 0.2	4.2 ± 0.1	4.5 ± 0.4	3.1 ± 0.9
Mushroom	2.4 ± 0.5	2.3 ± 1.0	3.9 ± 0.2	3.0 ± 0.5
Thyroid	1.4 ± 0.5	3.5 ± 0.3	2.6 ± 0.7	2.1 ± 1.0
Ionosphere	1.2 ± 0.3	2.9 ± 0.6	3.4 ± 0.3	1.6 ± 0.8
Soybean	3.1 ± 0.2	3.8 ± 1.5	5.0 ± 0.1	4.6 ± 0.3

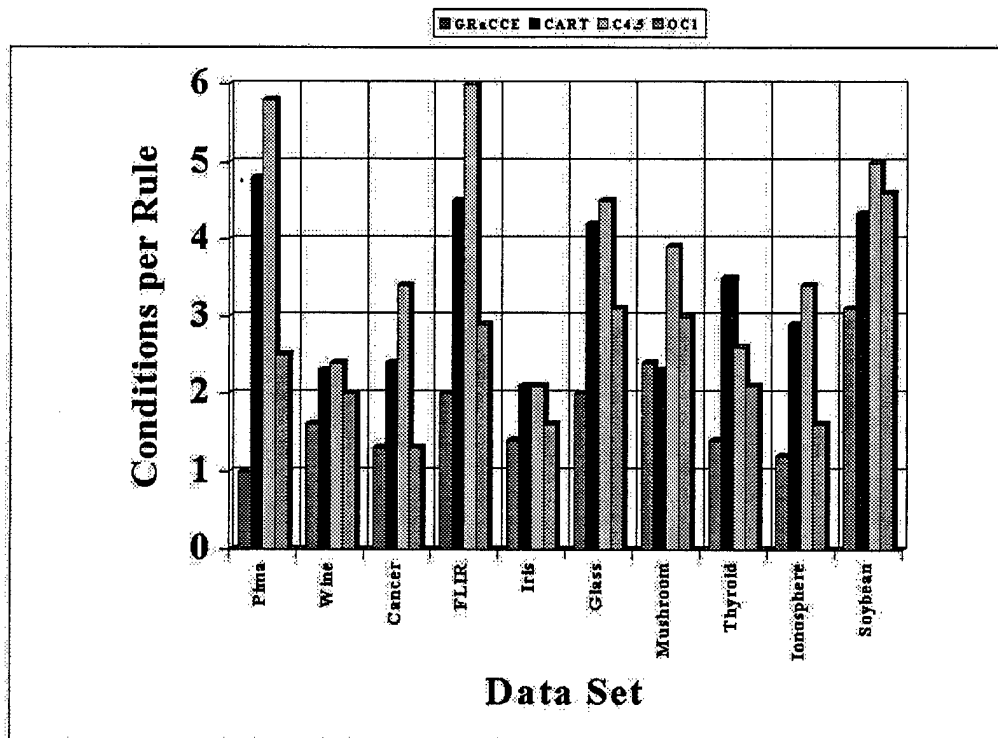


Figure 63 Baseline Comparison - Average Conditions per Rule for the Original Feature Set.

Data Set	Conditions per Rule ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	1.2 ± 0.5	4.1 ± 0.1	5.7 ± 0.2	1.9 ± 1.0
Wine	2.0 ± 0.5	2.1 ± 0.2	2.6 ± 0.3	1.6 ± 0.5
Cancer	1.1 ± 0.2	2.4 ± 0.2	3.3 ± 0.3	1.8 ± 1.0
FLIR	2.3 ± 0.6	4.5 ± 0.1	5.6 ± 0.3	3.1 ± 1.7
Iris	1.3 ± 0.0	2.3 ± 0.1	2.2 ± 0.3	1.6 ± 0.6
Glass	2.2 ± 0.3	3.2 ± 0.1	4.1 ± 0.3	3.2 ± 0.7
Mushroom	2.2 ± 0.5	2.7 ± 0.6	2.6 ± 0.0	3.5 ± 0.1
Thyroid	1.4 ± 0.3	3.4 ± 0.2	2.4 ± 0.8	2.5 ± 0.7
Ionosphere	1.7 ± 0.6	3.0 ± 0.2	3.5 ± 0.4	2.0 ± 0.9
Soybean	3.1 ± 0.7	3.8 ± 0.8	5.1 ± 0.1	4.3 ± 0.2

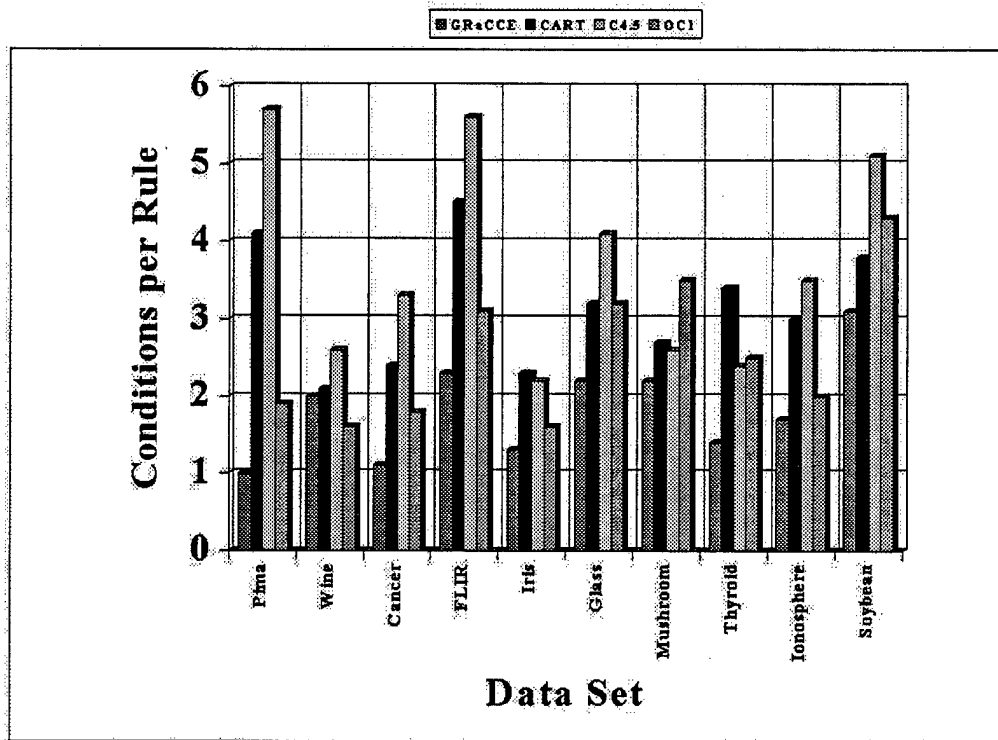


Figure 64 Baseline Comparison - Average Conditions per Rule for the Reduced Feature Set.

Data Set	Terms per Rule Set ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	15.4 \pm 2.7	1042.8 \pm 93.0	320.2 \pm 58.1	208.5 \pm 259.6
Wine	50.4 \pm 33.3	144.9 \pm 44.0	12.7 \pm 3.3	124.3 \pm 94.2
Cancer	18.6 \pm 8.9	85.6 \pm 21.5	37.6 \pm 14.4	33.0 \pm 29.3
FLIR	88.0 \pm 94.2	597.7 \pm 59.4	394.4 \pm 75.5	473.9 \pm 967.3
Iris	11.7 \pm 4.7	36.2 \pm 6.8	9.1 \pm 1.7	19.7 \pm 2.9
Glass	40.7 \pm 39.0	608.5 \pm 64.4	108.8 \pm 48.0	324.7 \pm 291.8
Mushroom	106.9 \pm 186.5	163.8 \pm 151.1	60.6 \pm 9.7	573.1 \pm 234.9
Thyroid	51.4 \pm 50.3	226.2 \pm 59.6	20.5 \pm 13.6	395.4 \pm 500.6
Ionosphere	76.5 \pm 57.5	619.7 \pm 41.0	37.2 \pm 11.7	251.1 \pm 277.1
Soybean	958.0 \pm 1056.6	1305.7 \pm 848.8	157.3 \pm 18.9	3947.7 \pm 69.2

Figure 65 Baseline Comparison - Average Terms per Rule Set (Original Feature Set).

Data Set	Terms per Rule Set ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	9.4 \pm 1.6	344.5 \pm 27.9	309.7 \pm 54.4	84.6 \pm 197.5
Wine	18.9 \pm 8.4	68.9 \pm 16.8	32.8 \pm 9.8	77.1 \pm 138.6
Cancer	15.0 \pm 7.3	85.6 \pm 21.5	37.6 \pm 14.4	33.0 \pm 29.3
FLIR	31.4 \pm 19.5	202.4 \pm 24.7	282.5 \pm 78.4	185.0 \pm 318.7
Iris	6.3 \pm 1.9	20.3 \pm 2.4	10.2 \pm 3.3	12.6 \pm 11.0
Glass	26.1 \pm 19.7	89.4 \pm 11.4	73.8 \pm 19.7	139.1 \pm 98.2
Mushroom	25.5 \pm 24.7	54.8 \pm 27.7	87.0 \pm 0.0	159.4 \pm 20.4
Thyroid	5.7 \pm 3.1	72.0 \pm 14.0	16.7 \pm 12.9	36.8 \pm 26.2
Ionosphere	93.3 \pm 77.4	294.7 \pm 8.7	41.7 \pm 16.4	174.3 \pm 180.6
Soybean	657.5 \pm 527.9	427.9 \pm 247.5	180.3 \pm 14.4	1334.8 \pm 16.9

Figure 66 Baseline Comparison - Average Terms per Rule Set (Reduced Feature Set).

Data Set	Rule Set Compactness ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	$2.0 \pm 0.0^{2,3,4}$	133.8 ± 11.9	320.2 ± 58.1	84.6 ± 32.4
Wine	$5.5 \pm 1.6^{2,3,4}$	11.2 ± 3.4	12.6 ± 3.3	9.6 ± 7.2
Cancer	$2.6 \pm 0.9^{2,3}$	13.0 ± 3.4	37.6 ± 14.4	3.7 ± 3.3
FLIR	$19.3 \pm 16.3^{2,3}$	100.6 ± 9.7	394.4 ± 75.5	79.0 ± 161.2
Iris	$4.0 \pm 0.2^{2,3,4}$	9.3 ± 1.8	9.1 ± 1.7	4.9 ± 0.7
Glass	$15.1 \pm 2.5^{2,3,4}$	78.2 ± 7.9	108.8 ± 48.0	38.1 ± 32.4
Mushroom	20.5 ± 7.9^3	17.2 ± 15.8	60.2 ± 9.7	26.1 ± 10.7
Thyroid	$4.1 \pm 2.6^{2,3,4}$	39.1 ± 11.0	20.5 ± 13.6	15.8 ± 20.0
Ionosphere	$2.7 \pm 1.4^{2,3,4}$	25.9 ± 12.4	37.2 ± 11.7	7.4 ± 8.2
Soybean	$64.3 \pm 3.4^{2,3,4}$	106.7 ± 70.4	157.3 ± 18.9	112.8 ± 35.9

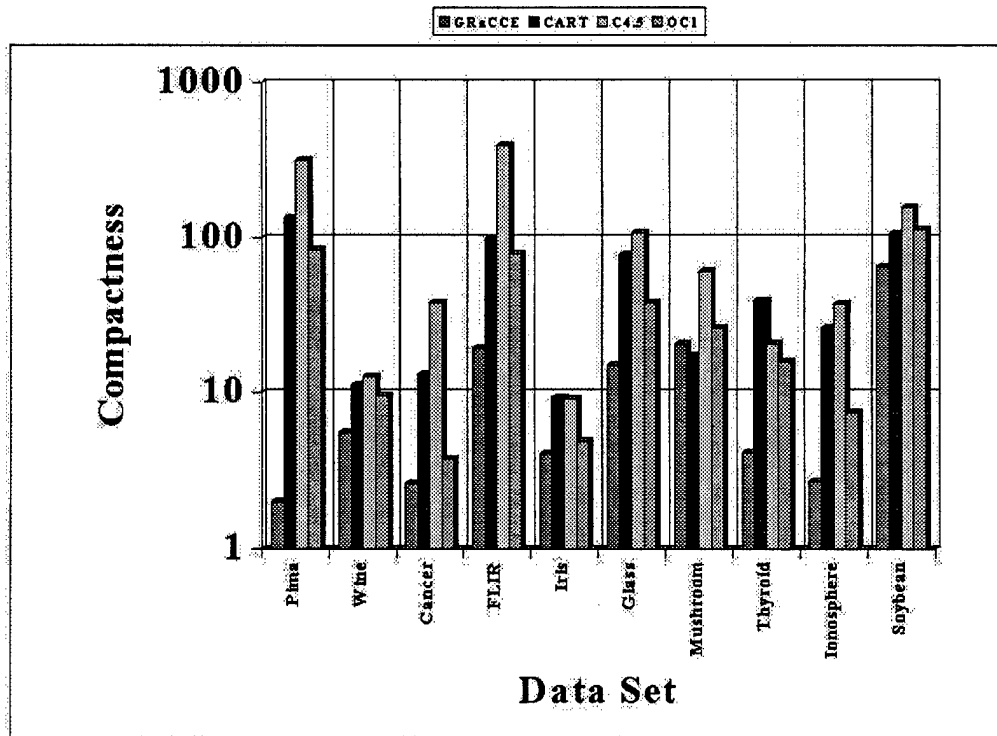


Figure 67 Baseline Comparison - Average Rule Set Compactness (Original Feature Set).

Data Set	Rule Set Compactness ($\mu \pm \sigma$)			
	Method			
	GRaCCE	CART	C4.5	OC1
Diabetes	$2.0 \pm 0.0^{2,3}$	69.8 ± 5.8	309.7 ± 54.4	16.9 ± 39.5
Wine	8.4 ± 2.5^3	9.6 ± 1.8	16.7 ± 5.6	6.1 ± 6.0
Cancer	$3.0 \pm 1.2^{2,3}$	13.0 ± 3.1	32.8 ± 9.8	12.9 ± 23.1
FLIR	$19.5 \pm 10.6^{2,3}$	104.1 ± 12.7	282.5 ± 78.4	92.5 ± 159.4
Iris	$4.0 \pm 0.0^{2,3}$	11.1 ± 1.3	10.2 ± 3.3	6.3 ± 5.5
Glass	$12.8 \pm 2.9^{2,3,4}$	29.0 ± 3.9	73.8 ± 19.7	34.8 ± 24.6
Mushroom	$13.3 \pm 4.5^{2,3,4}$	20.9 ± 10.3	87.0 ± 0.0	39.8 ± 5.1
Thyroid	$5.1 \pm 2.3^{2,3,4}$	36.7 ± 7.2	16.7 ± 12.9	18.4 ± 13.1
Ionosphere	$9.9 \pm 7.2^{2,3}$	25.5 ± 6.4	41.7 ± 16.4	12.5 ± 12.9
Soybean	$74.6 \pm 28.1^{3,4}$	67.3 ± 43.1	180.3 ± 14.4	89.0 ± 16.1

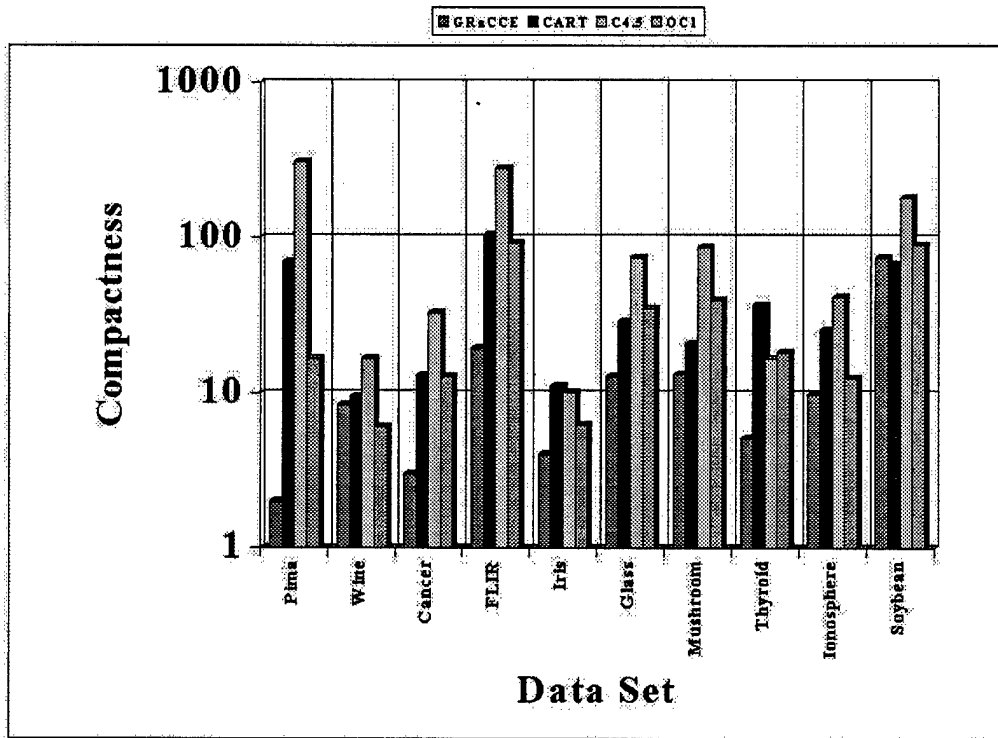


Figure 68 Baseline Comparison - Average Rule Set Compactness (Reduced Feature Set).

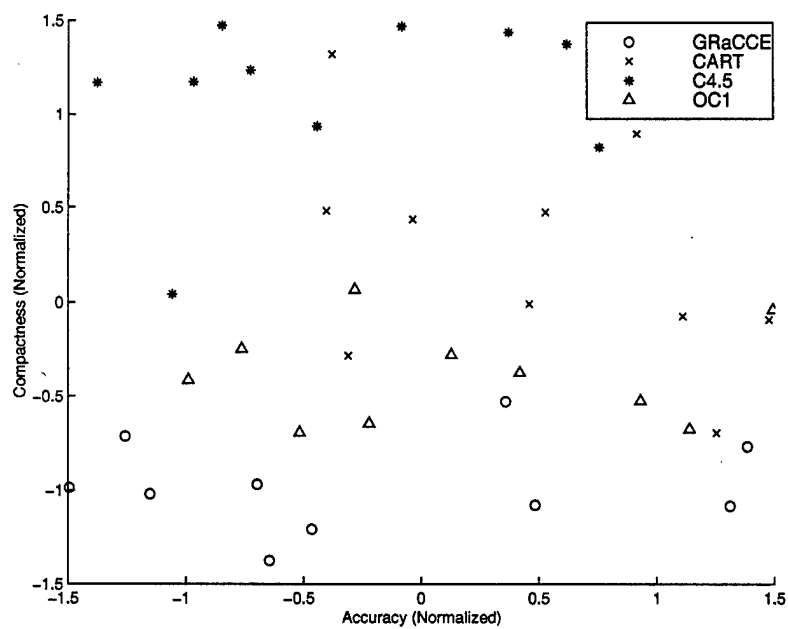


Figure 69 Normalized Comparison - Original Feature Set.

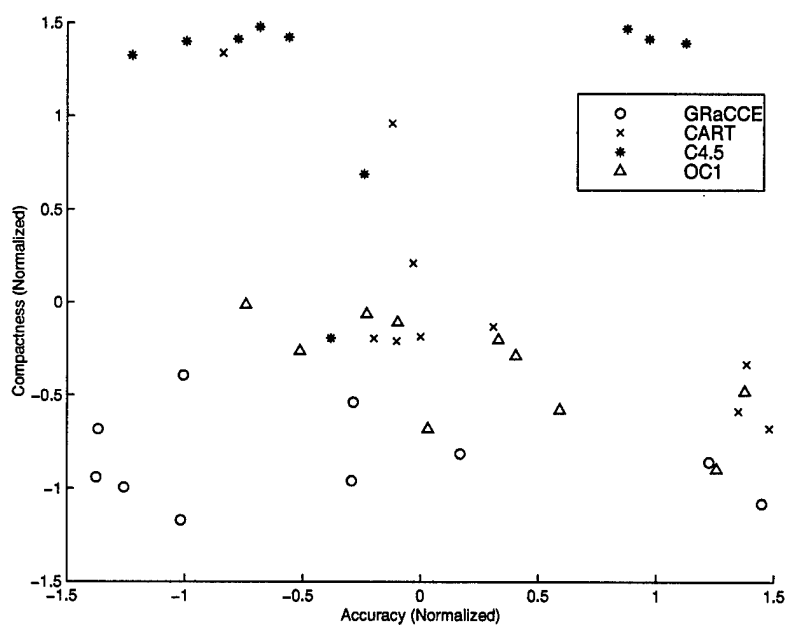


Figure 70 Normalized Comparison - Reduced Feature Set.

not always cover the whole data space; as such, orphan data (falling outside these regions) are automatically misclassified. As discussed in Section 4.9.2, it is possible to assign orphan data to the *closest* region. Region proximity is determined using the Mahalanobis distance metric (see Equation 20).

Figure 71 shows the degree to which this procedure improves the classification accuracy of each data set. For the most part, the reduction in error rate is incremental, falling in the 0 – 2% range. Although small, these improvements can be a significant proportion of the total error rate. Note that in cases where the decision rules covered the entire data space (such as Diabetes) no improvement occurred.

6.6.2 Effect of Partition Simplification. As stated earlier, the results presented in the inter-algorithm comparison are labeled *composite* because they include a mix of original and simplified rule sets; neither of these categories has a monopoly on the *best* results. This is because the partition simplification algorithm (PSA) sometimes results in an increased classification error rate. This assertion is supported by Figures 72 and 74, which show that the error rate increases as a result of the PSA in 75% of the cases examined. The difference in error is due to two main factors. The first is the settings of the PSA parameters (α and β); more liberal settings allow for greater differences in accuracy between the original and simplified rule sets. In addition, the use of an approximation of the data set (i.e., the weighted boundary points), in lieu of the entire training set, means that the error estimate is looser than it otherwise would have been. In light of these considerations, however, it is important to note that in most instances, the decrease in accuracy is small (under 3%). Given this, and keeping in mind the computational expense of the PSA, the decision to invoke this functionality should be based on the value placed on a simplified rule set by the user.

6.6.3 Robustness. Robustness refers to the extent to which a system's performance is dependent on its input parameters. If a small change to an input

Data Set	Error Rate ($\mu \pm \sigma$)	
	Feature Set	
	FULL	REDUCED
Diabetes	22.8 ± 2.5	22.8 ± 2.2
Wine	4.5 ± 2.7	7.1 ± 3.4
Cancer	3.7 ± 1.6	4.5 ± 2.0
FLIR	24.2 ± 2.8	22.8 ± 3.6
Iris	2.8 ± 2.6	3.0 ± 2.6
Glass	35.4 ± 6.3	32.5 ± 5.8
Mushroom	6.5 ± 2.6	2.1 ± 1.7
Thyroid	1.9 ± 0.7	1.9 ± 0.9
Ionosphere	9.4 ± 2.3	12.5 ± 6.8
Soybean	9.0 ± 2.1	14.4 ± 5.3

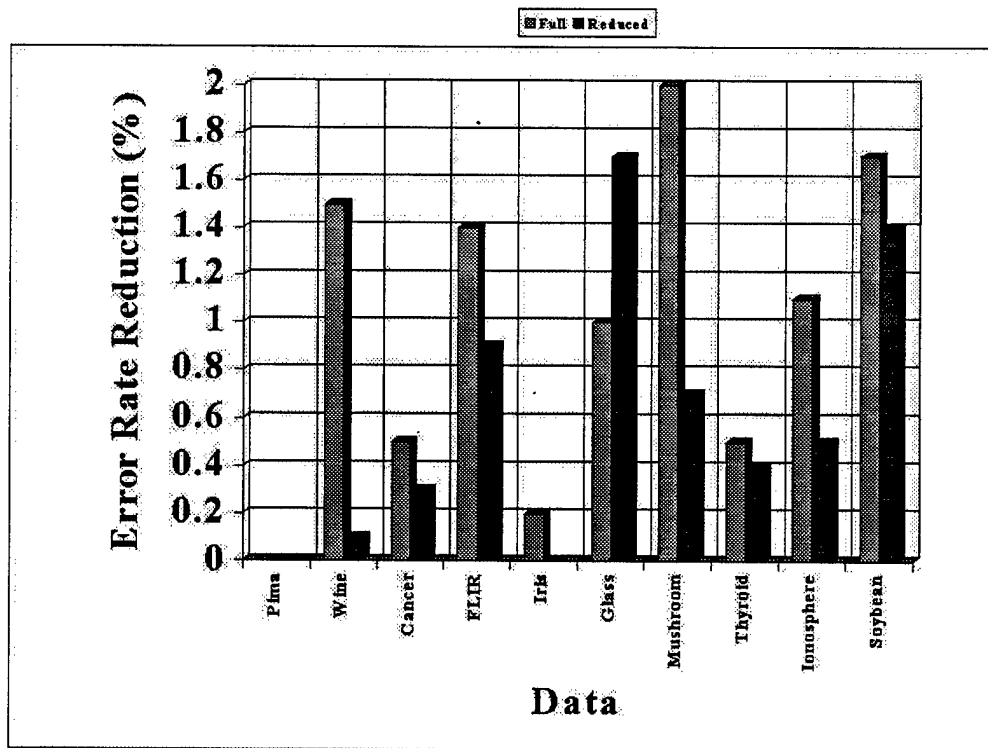


Figure 71 This graph shows the mean error rate reduction (relative to the rules-only results) when orphan data is classified using the Mahalanobis distance metric.

Data Set	Error Rate ($\mu \pm \sigma$)		
	Original	Simplified	Δ
Diabetes	22.8 ± 2.5	26.2 ± 3.7	+3.4
Wine	6.6 ± 3.4	10.2 ± 5.0	+3.6
Cancer	4.4 ± 1.7	5.2 ± 1.9	+0.8
FLIR	26.2 ± 2.9	27.3 ± 2.4	+1.1
Iris	3.2 ± 4.1	5.1 ± 4.0	+1.9
Glass	42.4 ± 9.4	37.3 ± 7.0	-5.1
Mushroom	13.4 ± 5.4	8.6 ± 3.3	-4.8
Thyroid	2.8 ± 1.4	10.2 ± 22.9	+7.4
Ionosphere	11.3 ± 2.8	13.3 ± 4.6	+2.0
Soybean	12.4 ± 3.4	12.4 ± 4.5	+0.0

Figure 72 Error Rate Changes Resulting from Partition Simplification (Original Feature Set).

Data Set	Terms per Rule Set ($\mu \pm \sigma$)		
	Original	Simplified	DPS
Diabetes	16.0 ± 0.0	6.1 ± 4.5	0.62 ± 0.28
Wine	70.9 ± 20.4	15.4 ± 10.1	0.77 ± 0.20
Cancer	23.4 ± 7.9	10.6 ± 4.0	0.54 ± 0.15
FLIR	115.7 ± 98.0	36.7 ± 29.4	0.63 ± 0.17
Iris	16.2 ± 0.9	7.6 ± 1.6	0.53 ± 0.12
Glass	135.9 ± 22.4	24.2 ± 5.9	0.82 ± 0.03
Mushroom	449.9 ± 172.8	40.3 ± 18.6	0.91 ± 0.02
Thyroid	103.1 ± 64.9	14.7 ± 13.6	0.81 ± 0.28
Ionosphere	91.2 ± 46.9	25.5 ± 11.2	0.68 ± 0.19
Soybean	2251.6 ± 120.0	119.3 ± 13.4	0.95 ± 0.01

Figure 73 Reduction in Rule Set Complexity resulting from Partition Simplification (Original Feature Set).

Data Set	Error Rate ($\mu \pm \sigma$)		
	Original	Simplified	Δ
Diabetes	22.8 ± 2.2	26.7 ± 3.4	+3.9
Wine	8.1 ± 3.4	8.8 ± 4.8	+0.7
Cancer	4.6 ± 1.9	6.4 ± 2.2	+1.8
FLIR	24.3 ± 3.3	24.0 ± 3.6	+0.3
Iris	3.0 ± 2.6	5.5 ± 3.5	+2.5
Glass	40.4 ± 6.8	36.1 ± 6.2	-4.3
Mushroom	4.2 ± 2.9	4.1 ± 4.0	-0.1
Thyroid	3.0 ± 1.8	2.3 ± 1.0	-0.7
Ionosphere	14.1 ± 6.8	15.9 ± 6.1	+1.8
Soybean	16.4 ± 5.1	19.3 ± 7.2	+2.9

Figure 74 Error Rate Changes Resulting from Partition Simplification (Reduced Feature Set).

Data Set	Terms per Rule Set ($\mu \pm \sigma$)		
	Original	Simplified	DPS
Diabetes	10.0 ± 0.0	5.4 ± 2.2	0.46 ± 0.22
Wine	25.3 ± 7.4	13.6 ± 5.9	0.47 ± 0.11
Cancer	18.0 ± 7.3	7.3 ± 4.0	0.60 ± 0.12
FLIR	39.0 ± 21.3	26.9 ± 13.3	0.29 ± 0.14
Iris	8.0 ± 0.0	4.3 ± 0.7	0.46 ± 0.09
Glass	51.2 ± 11.5	16.6 ± 5.3	0.68 ± 0.06
Mushroom	53.4 ± 18.2	15.8 ± 6.5	0.71 ± 0.04
Thyroid	10.2 ± 4.5	5.7 ± 3.1	0.45 ± 0.09
Ionosphere	138.8 ± 100.5	29.4 ± 22.2	0.76 ± 0.10
Soybean	1119.0 ± 421.4	112.6 ± 36.4	0.90 ± 0.01

Figure 75 Reduction in Rule Set Complexity resulting from Partition Simplification (Reduced Feature Set).

parameter causes a large difference in performance, the system is considered brittle with respect to that parameter. Of course, it is preferable to use systems that exhibit incremental differences in performance in response to like changes in parameter values. Given this, it is of interest to analyze GRaCCE's performance in this respect.

Because its fundamental task is to find class homogeneous regions in data, GRaCCE's robustness is predicated on its ability to cluster data. While there is no shortage of clustering algorithms, many are heavily dependent on user provided input parameters⁴. Specifically, an algorithm tends to produce optimal results if the user knows the required information ahead of time; conversely, the results can be very poor if the user guesses wrong. This lack of *robustness* of such algorithms is a problem as the user rarely has enough insight about the structure of the data to provide the best set of parameters. According to Jain and Dubes (65), there is no single best criteria for judging a cluster because no precise and workable definition of "cluster" exists. For the purpose of this test, however, cluster quality is estimated with respect to two metrics: classification accuracy and the number of clusters. These metrics are based on the premise that a good clustering algorithm should maximize classification accuracy with the fewest possible clusters.

Based on the above discussion, robustness is determined by measuring how sensitive GRaCCE's performance is (in terms of the metrics discussed above) to changes in the cluster purity parameter (γ_{min}). This parameter is chosen as the basis for this experiment because it has the most influence on how solutions are evaluated by the objective function (Φ), refer to Equations 16 thru 18). In order to evaluate performance changes, the γ_{min} levels were varied from 0.7 to 1.0. These values represent the most reasonable levels of cluster purity that a user might specify. In short, γ_{min} reflects the user's preference rather than his guess about the clustering structure. The data sets evaluated for this exercise are: Iris, Cancer, Wine, Glass, and Syn04. The Syn04 data set was selected due to its larger number of modes (25).

⁴A comparison of GRaCCE to other clustering techniques can be found in (90).

This experiment revealed GRaCCE to be more robust than expected. The results for each data set are shown in Figures 76 through 80. While the results differ for each data set tested, only one trend is evident: as the required level of purity decreases, the number of clusters generated also decreases. This effect was anticipated as requiring high levels of purity will naturally cause cluster fragmentation. While this was anticipated, the real surprise was the absence of a corresponding degradation in classification accuracy. It was expected that allowing greater impurity would result in a much higher error rate. Although this did occur for the SYN04 data set, for the other data sets the error rate was relatively constant or even *decreased*.

One explanation for these unexpected results is that the objective function favors regions containing large clusters (of the target class) bounded by a minimum of partitions, as long as these meet the γ_{min} purity threshold. By decreasing γ_{min} , we increase the fitness of larger, but less pure, clusters; thus, it is more likely to retain such solutions in the population. While this strategy may degrade accuracy in the short term, it also affords the opportunity to evolve variations of these clusters that are more fit with respect to purity.

This effect may not extend to highly multi-modal data sets. The lower modality of most data sets gave them a propensity to retain the natural cluster structure. When a data set has a large number of modes per class (as SYN04 does), the natural cluster structure is harder to retain as purity drops. Eventually, a point is reached where relaxing the purity requirement has a disastrous effect on accuracy. For SYN04, this level was $\gamma_{min} = 0.8$; for data sets with fewer modes, the threshold is probably lower. Thus, the purity level selected should be based on the number of modes per class.

6.7 Assessing cGRaCCE

The last test objective is to demonstrate cGRaCCE, which implements a concurrent version of the RI phase (88). Since cGRaCCE only implements a single phase

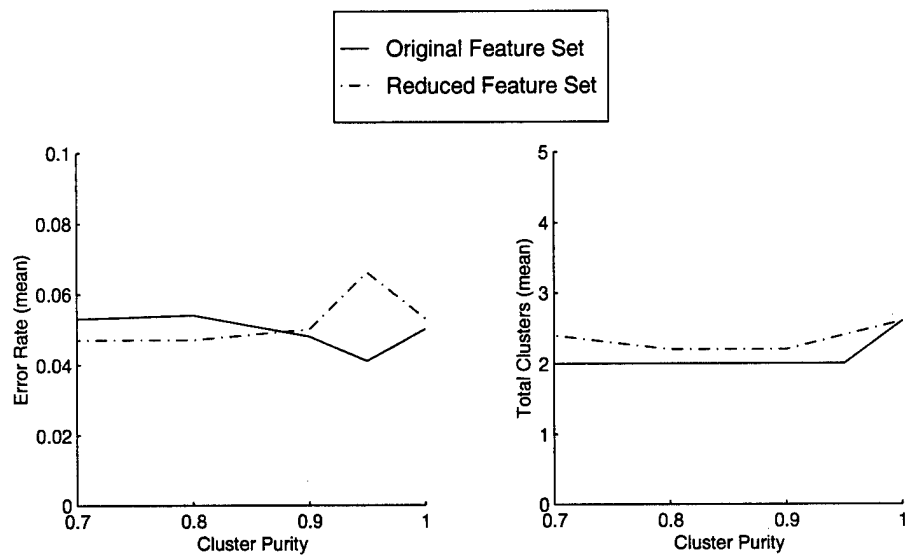


Figure 76 Effect of γ_{min} on Cancer Data.

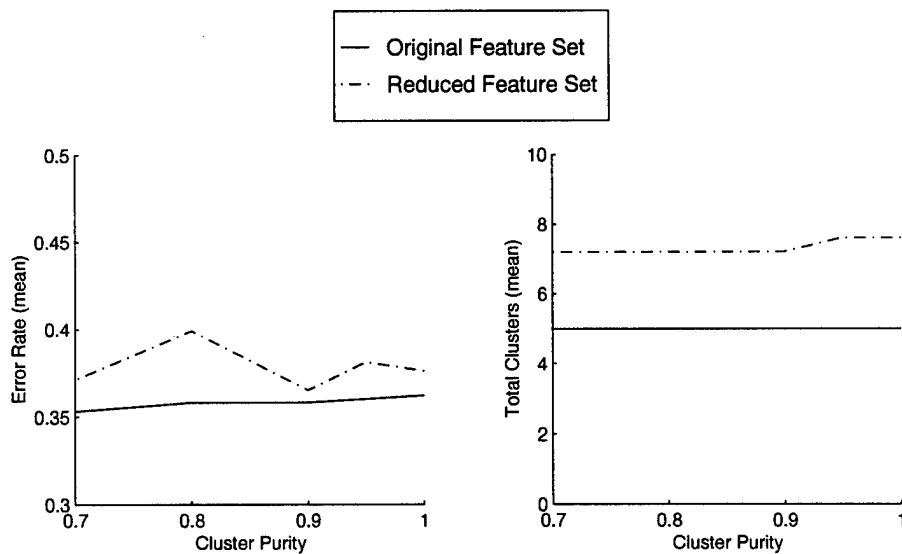


Figure 77 Effect of γ_{min} on Glass Data.

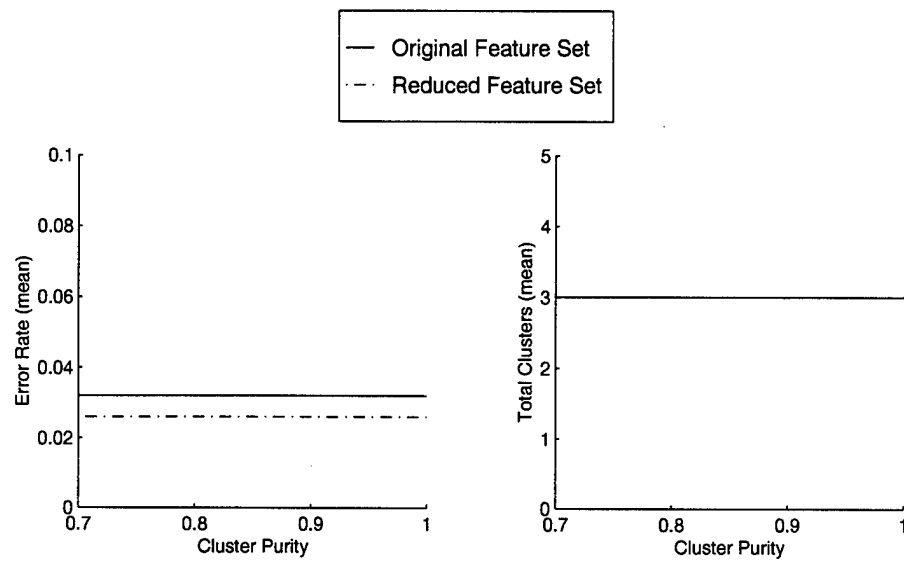


Figure 78 Effect of γ_{min} on Iris Data.

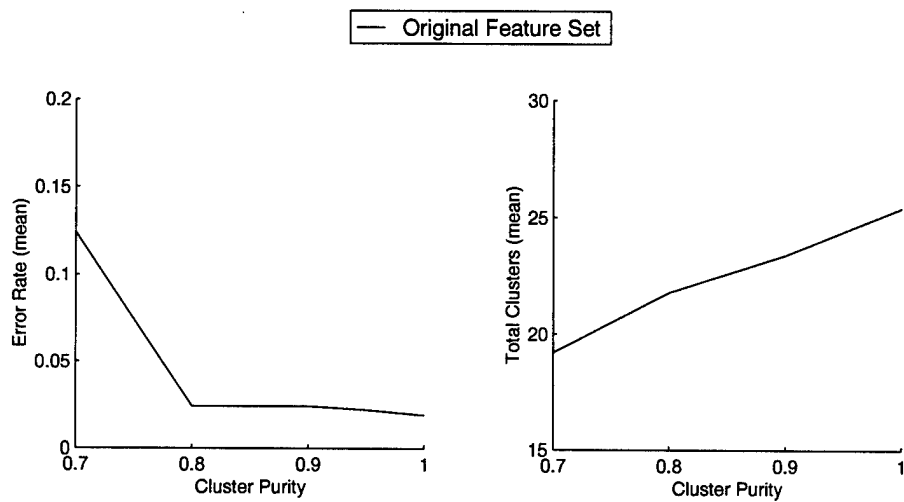


Figure 79 Effect of γ_{min} on Syn04 Data.

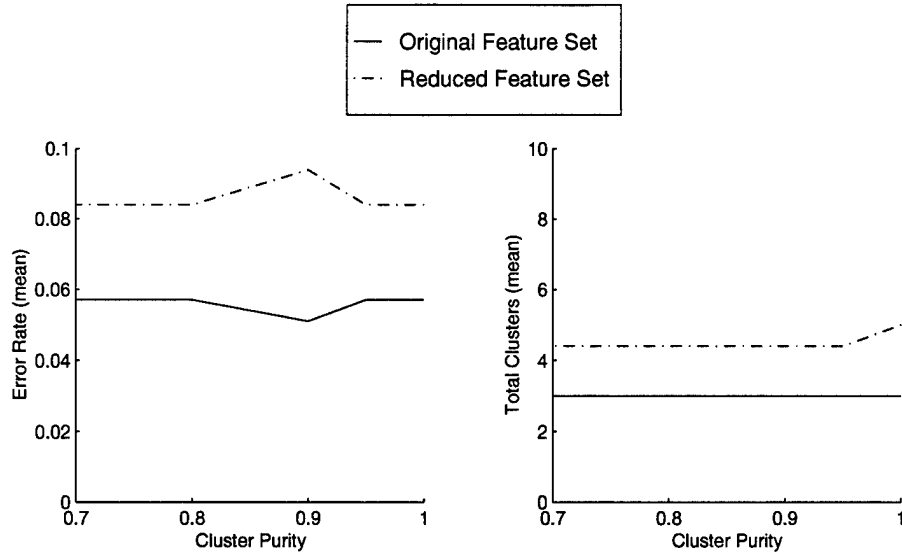


Figure 80 Effect of γ_{min} on Wine Data.

of the algorithm, a direct comparison of its run-time performance against the decision tree algorithms is not accomplished. Likewise, comparing it to the full-scale version of GRaCCE would also be of little value as they both use the same algorithm. Because the primary motivation behind cGRaCCE was to determine if GRaCCE could be successfully parallelized, the focus of the testing is to assess cGRaCCE with respect to this goal. In particular, this involves testing how the the execution speed of cGRaCCE improves as it is run on an ever more distributed environment. In addition, the scalability of the RI phase for different data sets is examined. To meet these sub-objectives, cGRaCCE's run-time performance is measured for data sets that vary in terms of the following characteristics:

- Number of classes.
- Number of modes per class (i.e, unimodal versus multi-modal data).
- Number of boundary points.
- Distribution of boundary points among classes. The balance factor (BF) metric in Equation 47 provides an estimate of this distribution. The closer the BF is

Table 24 Data Set Descriptions.

Data Set	Classes	Features (Original/Used)	Modes	Boundary Points	Balance Factor
Cancer	2	9/6	2	17	0.029
Glass	6	9/4	6	23	0.138
Mushroom	2	22/4	2	39	0.064
Soybean	19	19/15	19	103	0.023
SYN03	5	2/2	5	61	0.062
SYN04	2	2/2	25	105	0.020
Wine	3	13/3	3	21	0.032

to 1, the more uneven the distribution of boundary points among the classes; a BF of 0 indicates a perfectly balanced distribution.

$$\text{Balance Factor} = \frac{1}{m} \sum_{i=1}^m \left(\frac{|\lambda_i|}{|\Lambda|} - \frac{1}{m} \right) \quad (47)$$

With these requirements in mind, the data sets selected for testing are described in Table 24. Two of these are synthetic data sets specifically designed to exercise different aspects of the cGRaCCE algorithm. In particular, SYN04 should require many CH region searches (due to its highly multi-modal nature). In contrast, SYN03 has more classes (5), but should require only one search per class since each class is unimodal. The Soybean data set was used to test performance when a large number of classes (19) is involved. The remaining data sets were chosen because they each have a small number of classes. These totals are low enough for each class to have a dedicated processor on the PC cluster; as a result, all classes can be processed in parallel.

The execution times of each of the data sets are measured for the region identification phase over several different test cases. Each test case is defined by two basic parameters: the hardware configuration (in terms of the number of processors utilized) and the size of the convergence window (q) as defined in Figure 32. The

results reported for each test case are averaged over a series of thirty runs. The test cases constructed for each data set were tailored to its characteristics. For a given data set, the number of processors utilized is limited either by the constraints of the available hardware or by the number of data classes. For example, since the Wine data set has three classes, a maximum of three processors are used.

6.7.1 Effect of Concurrency on Run-Time Execution Performance. The results of these tests in terms of execution time and speed-up (relative to the single processor configuration) are documented in Figures 81 through 87. While using a multi-processor configuration almost always resulted in some degree of speed-up, the overall effect was less substantial than expected. In general, the speed-up resulting from adding additional processors was sub-linear (i.e, less than m); the only case where it approached this ideal was for the SYN04 data set. Indeed, slow downs actually occurred for the Wine and Cancer data sets when the convergence window was set to its minimum ($q = 10$). For these cases, the duration of each CH region search is short (≈ 0.2 sec) as compared to SYN04 (which has a duration of 3 sec or greater).

An in-depth analysis of the raw data revealed several reasons for these uneven results. Perhaps the major contributing factor is that the instruction cache on each processor is only loaded *after* the first CH search has completed. Consequently, the first CH search on each processor takes up to $2\times$ longer than if it had not been elaborated as a separate task. This situation leads to sub-linear speed-up on data sets like SYN03 which have many classes, but result in few CH searches per processor. In contrast, SYN04 exhibits nearly linear speed-up because sufficient CH searches are executed on each processor to overcome this initial disadvantage.

Interprocessor communication (IC) overhead also impacted the observed speed-up rates. Although all tasks are spawned simultaneously, MPI uses a binary tree structure to broadcast shared data. Starting with the root task, the shared data is

passed as required to up to two child tasks per parent. As a result, increasing the number of tasks lengthens the broadcast path. Overall, a logarithmic increase in the IC overhead (which averages 0.12 sec per inter-task broadcast) occurs as the number of classes grows. This overhead is particularly expensive for tasks with execution times of 1.0 sec or less; this is frequently the case for low values of q (i.e., $q \leq 20$). In contrast, IC overhead is a much lower percentage of total duration for tasks with large q values. This explains why speed-up generally improves as q increases. While large imbalances in each processor's workload can also degrade the speedup rate, the relatively small balance factors in Table 24 indicate this is a minor issue.

The above findings indicate that cGRaCCE, in its current form, most improves the run-time execution performance of highly multi-modal data sets. Because data sets with this characteristic require a large number of searches per processor, they are less impacted by instruction cache latency and IC overhead. In contrast, little (or negative) benefit will accrue to data sets that have a small number of classes and unimodal class structures (as they are most affected by these factors). As the number of classes increases, however, the speed-up rate will improve provided each class can be assigned its own processor⁵. While the extra processors can partially compensate for the instruction cache latency and IC overhead, the speed-up rate remains sub-linear.

6.7.2 Scalability. Recall that the *worst case* algorithmic growth rate for the region identification phase was derived for both GRaCCE and cGRaCCE. in the previous chapter. Nonetheless, the key question left unanswered is: how well does GRaCCE's run-time execution performance scale in practice? Using the data collected for these experiments, we try to shed some light on this issue.

⁵Although the performance of the Soybean data set catastrophically degrades when the number of processors is increased to 11, the cause of this was traced to a problem with MPI which should be corrected in future versions

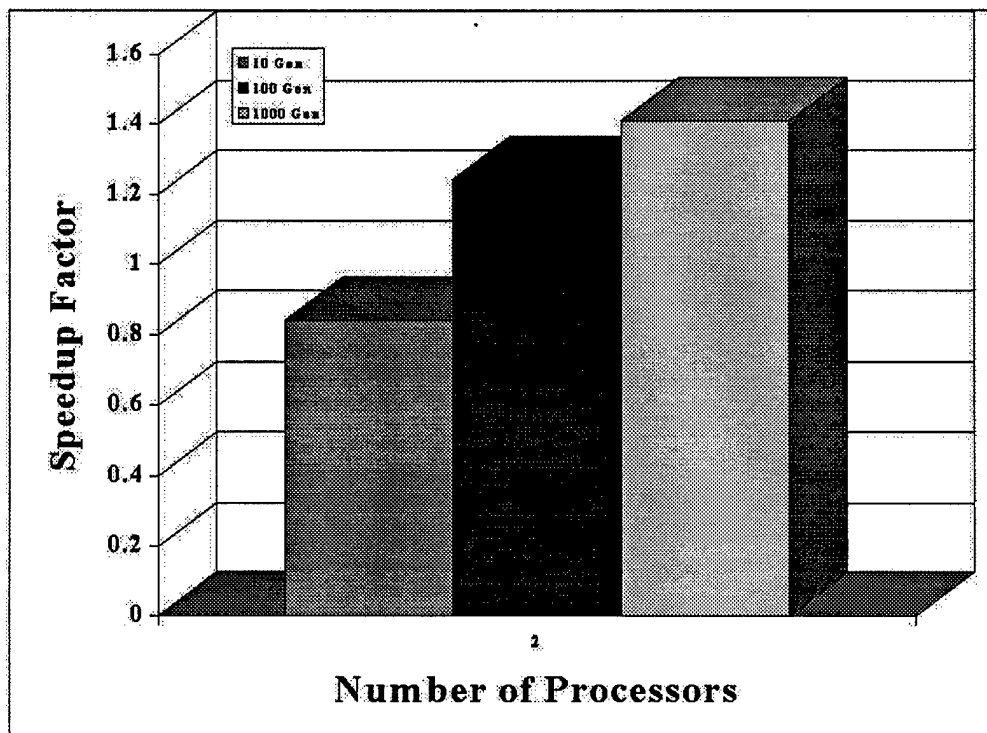
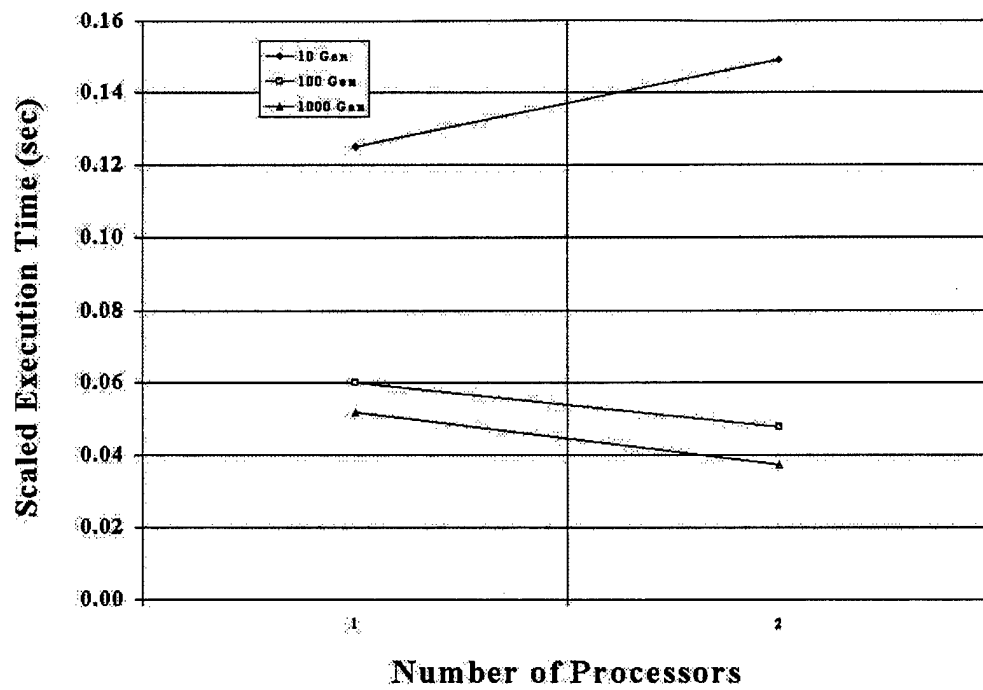


Figure 81 Effect of Concurrency - Cancer Data.

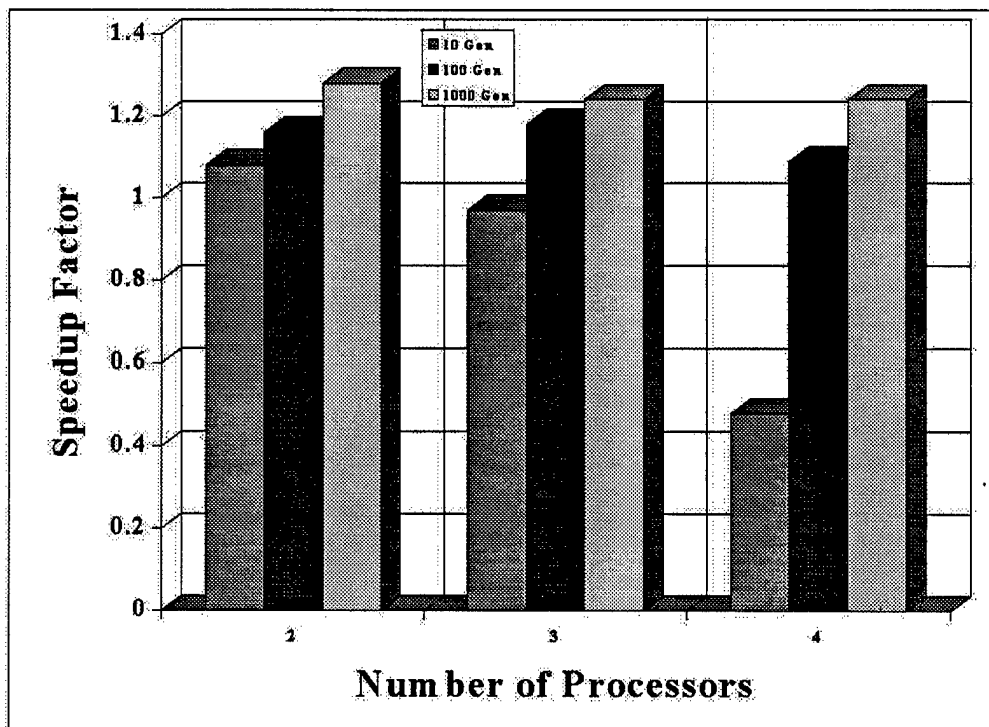
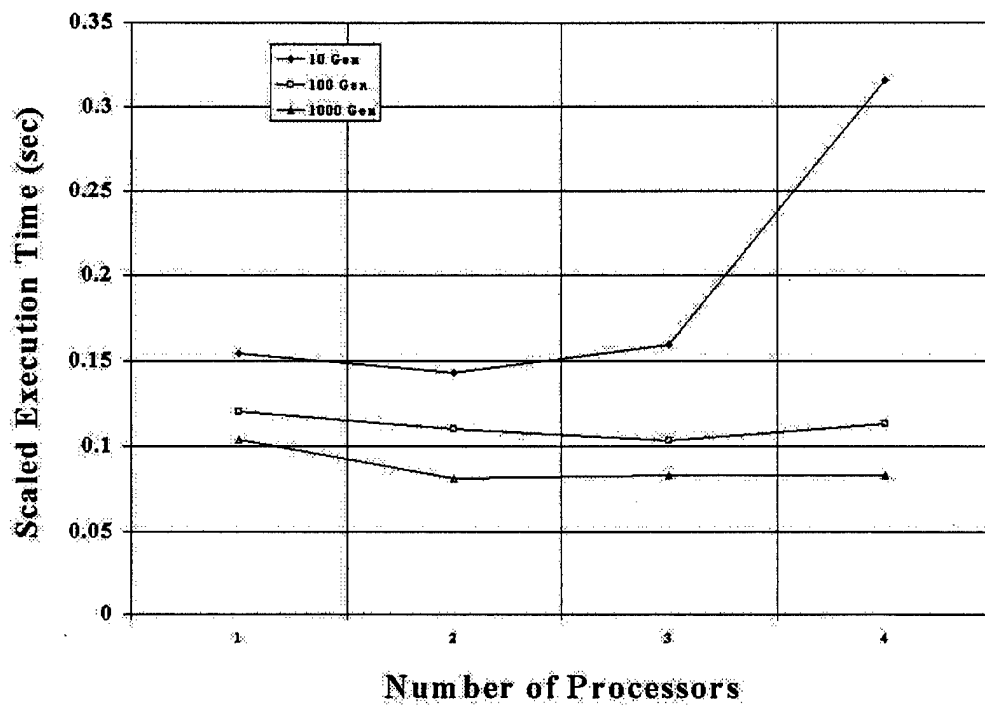


Figure 82 Effect of Concurrency - Glass Data.

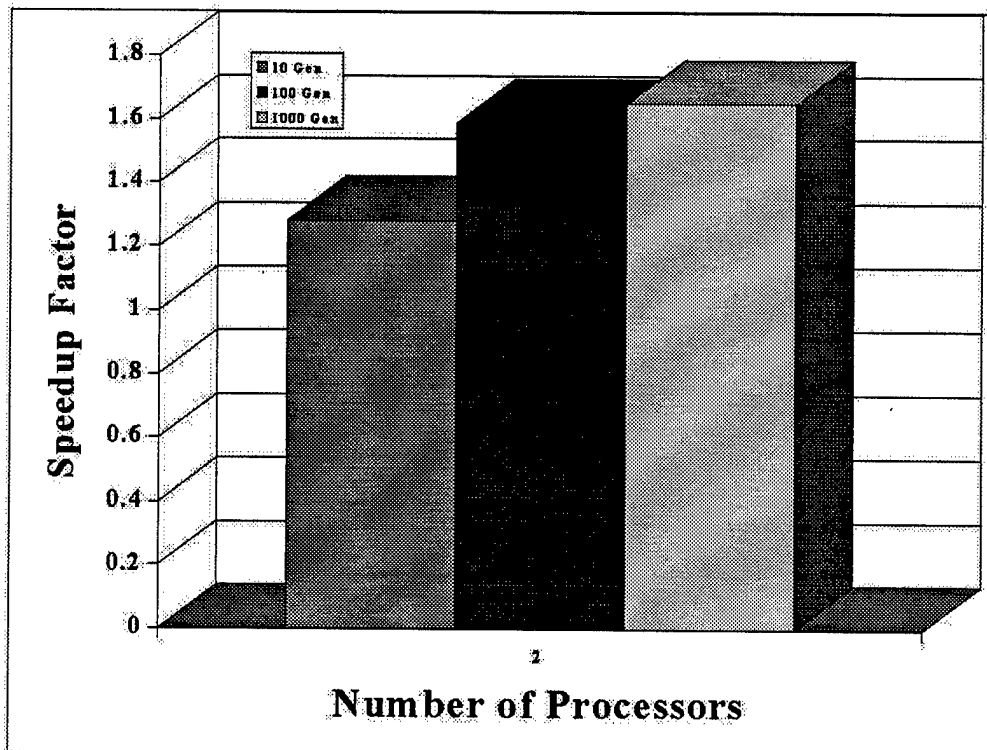
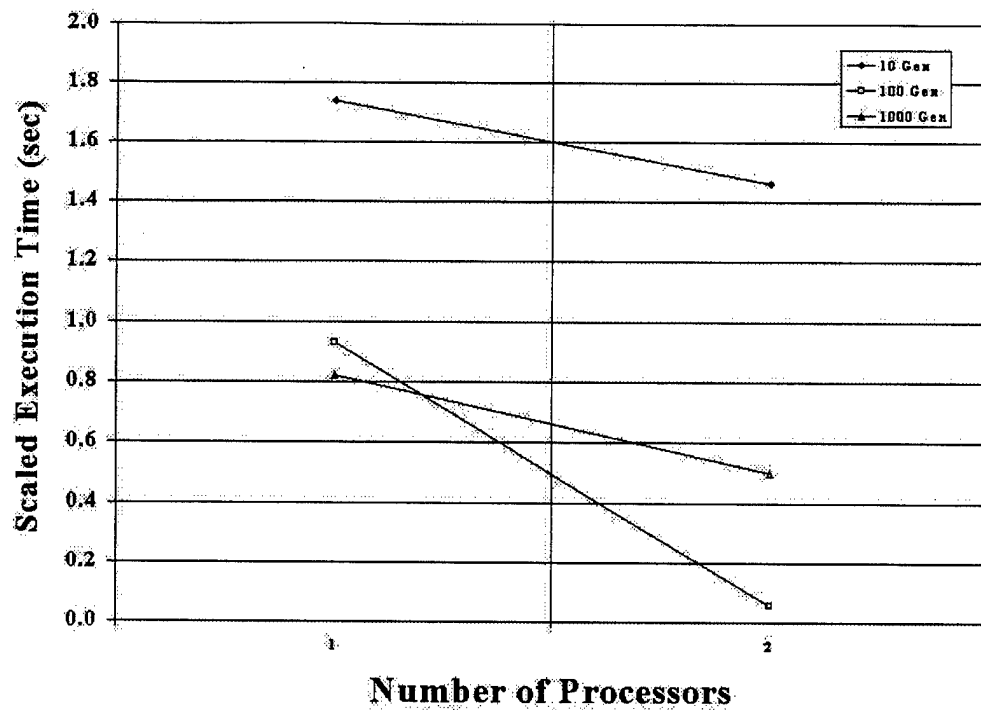


Figure 83 Effect of Concurrency - Mushroom Data.

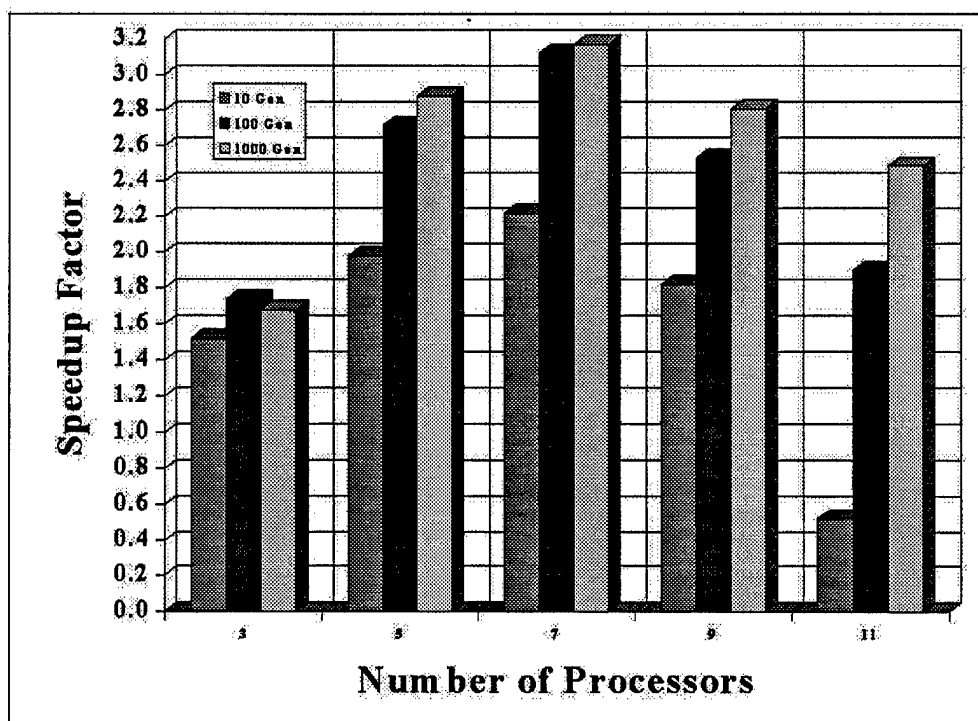
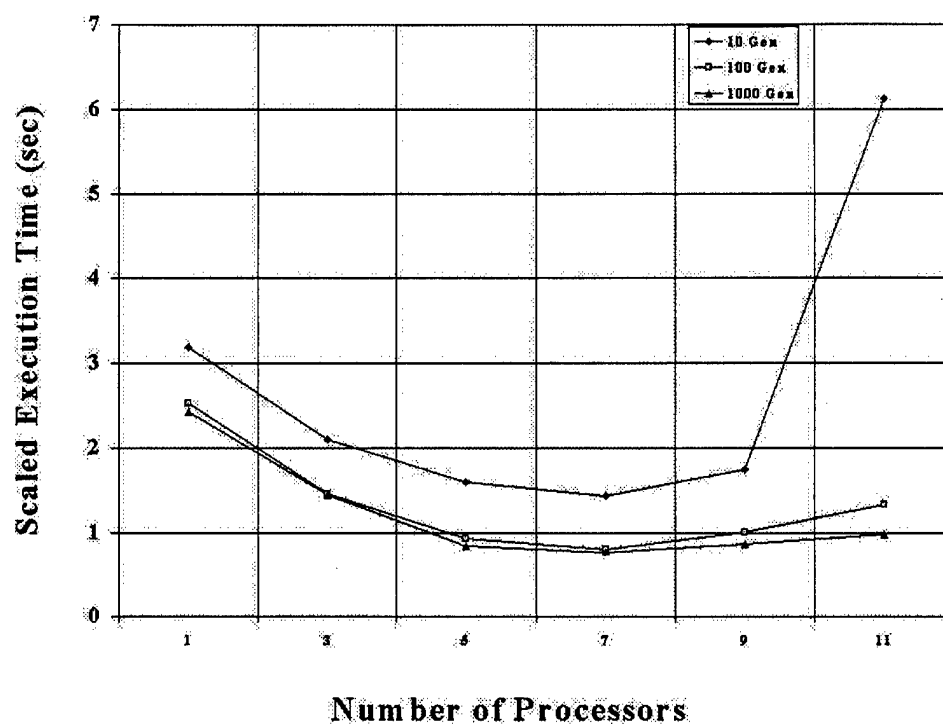


Figure 84 Effect of Concurrency - Soybean Data.

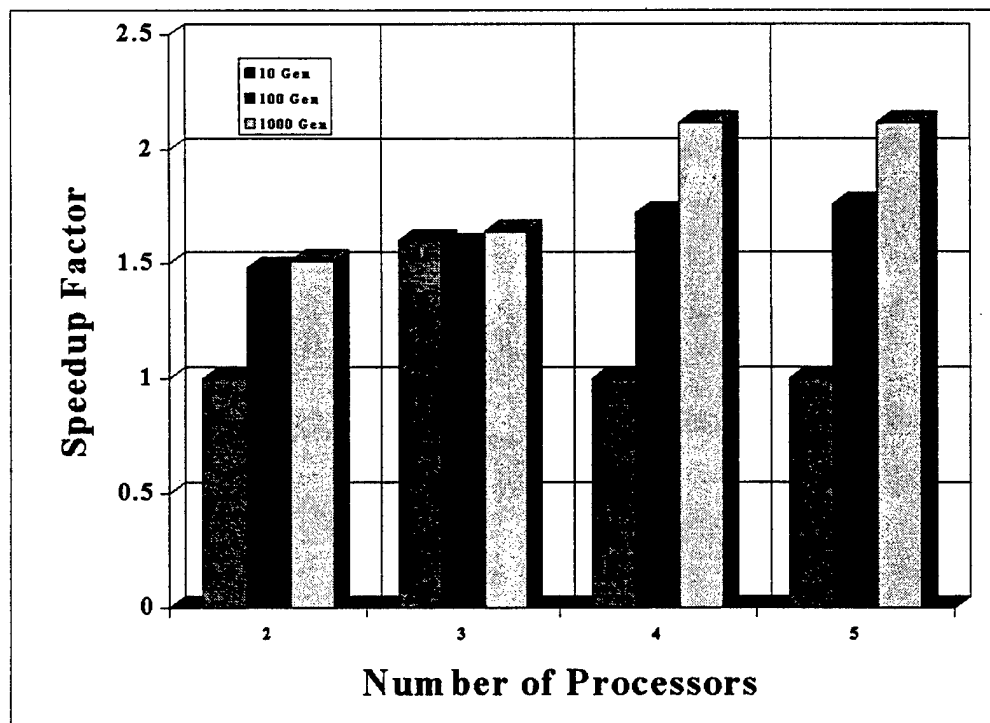
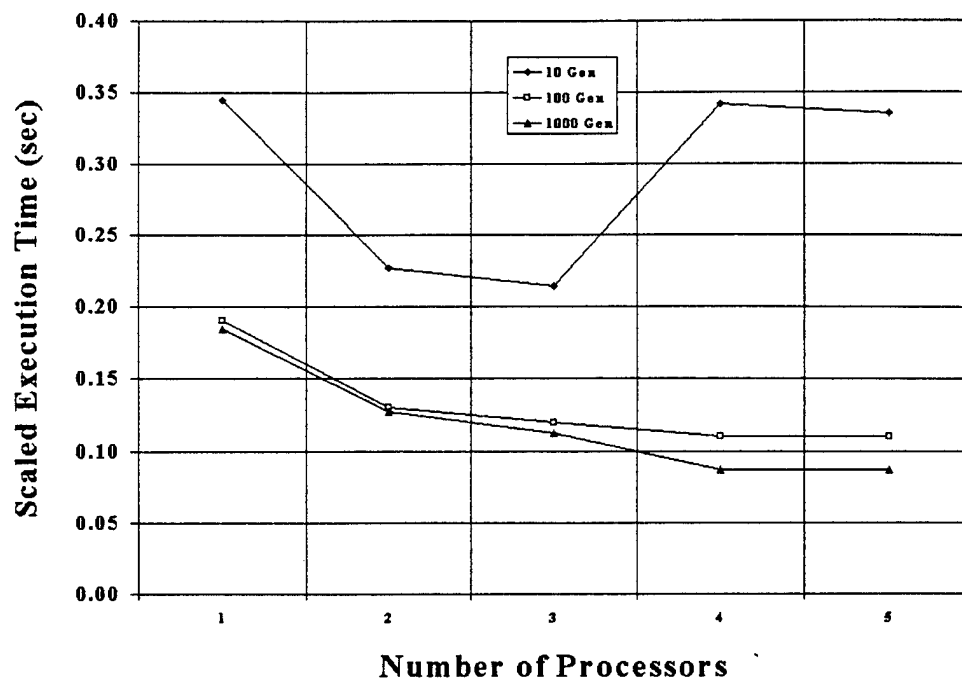


Figure 85 Effect of Concurrency - SYN03 Data.

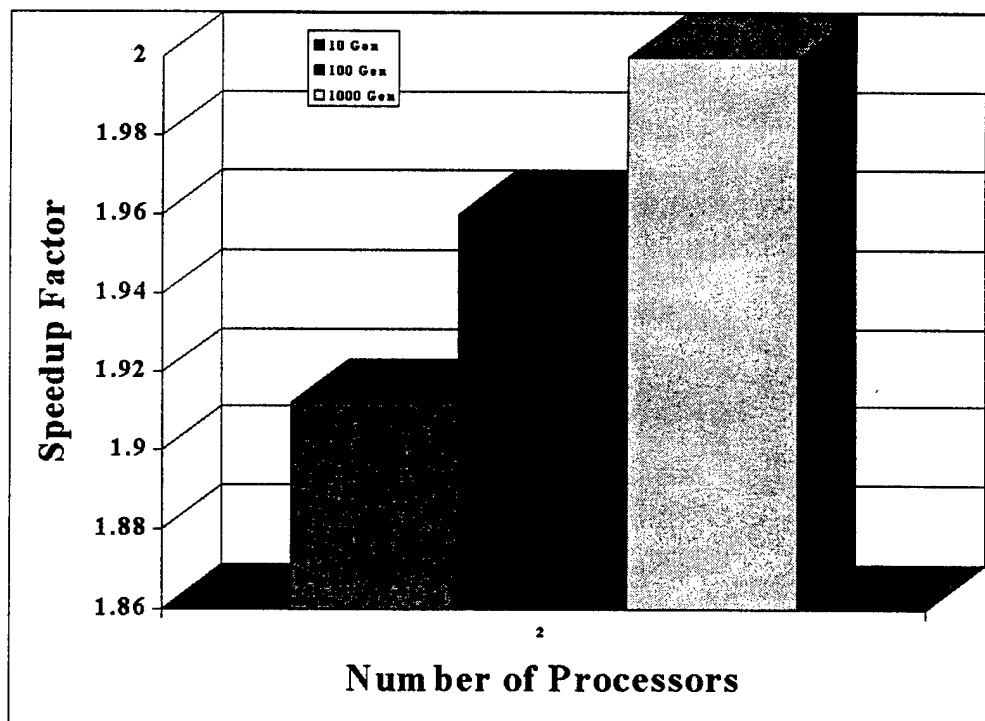
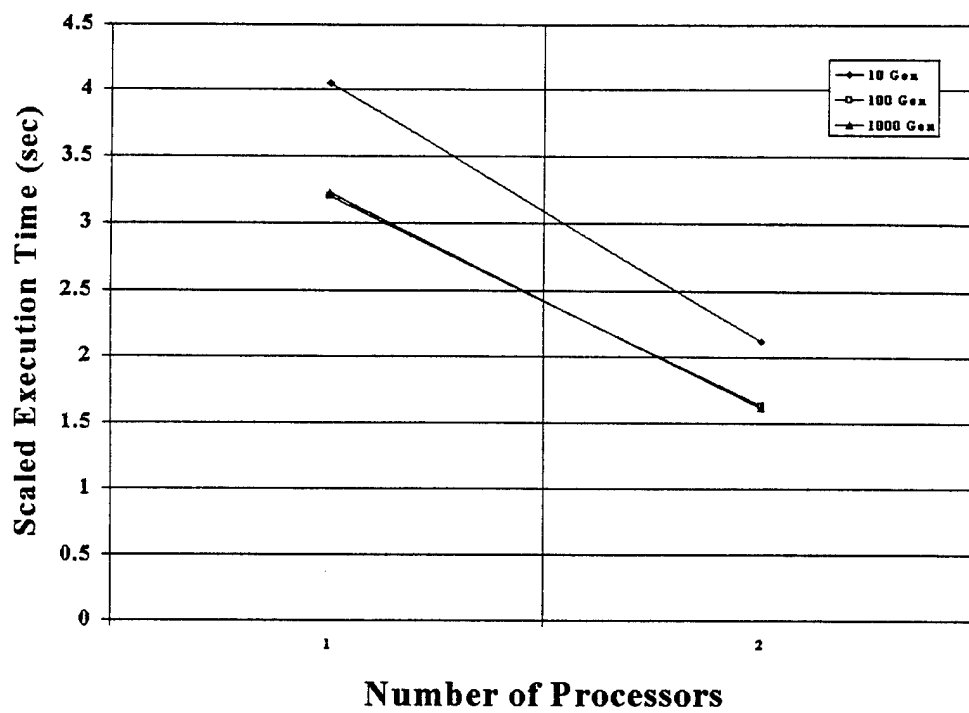


Figure 86 Effect of Concurrency - SYN04 Data.

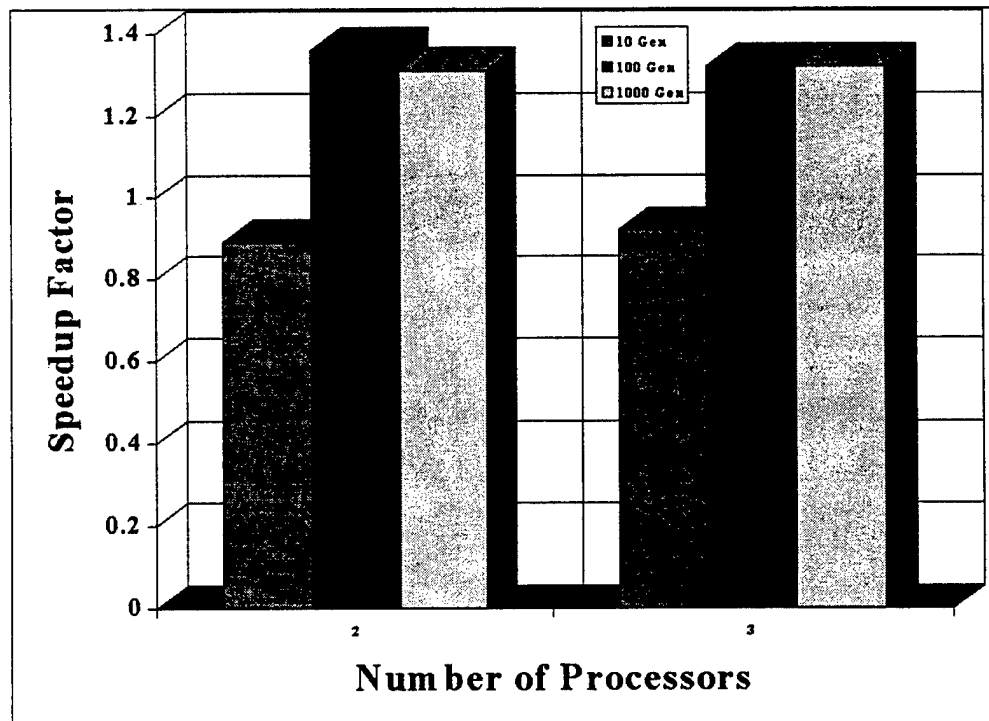
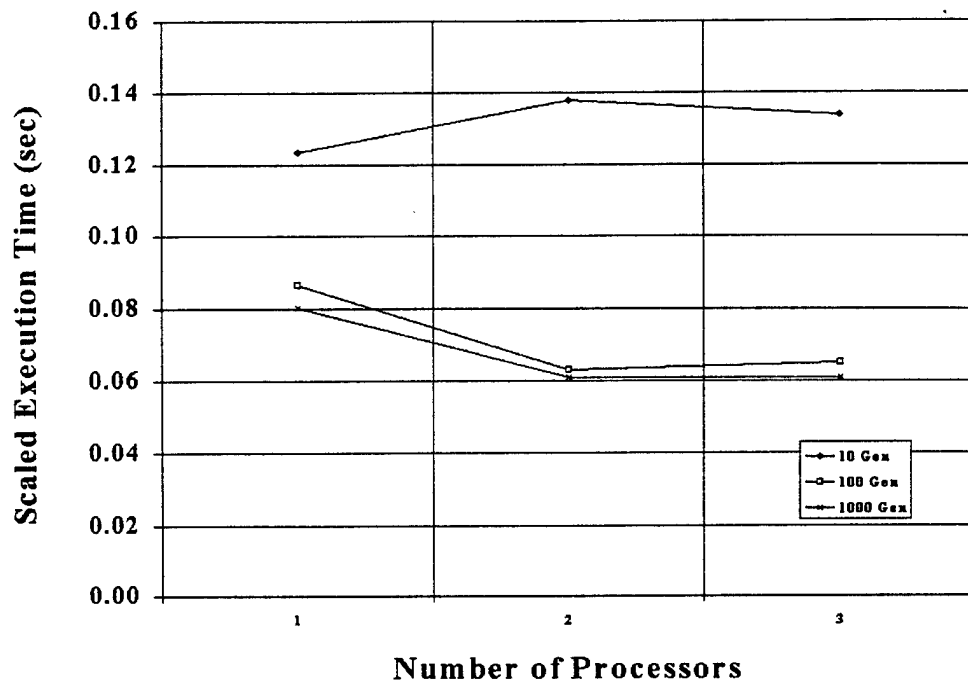


Figure 87 Effect of Concurrency - Wine Data.

A data mining algorithm can be analyzed in terms of how its execution time increases as the data set grows in complexity or size. With regard to the former criteria, a visual representation of the system's scalability is obtained by plotting its execution time versus the number of boundary points in the data set. Studies of data mining algorithms have traditionally used data set size as the basis for measuring run-time efficiency (1; 132; 154); data set complexity was largely ignored (perhaps because it is much harder to gauge). As was demonstrated in Section 5.5, the structural complexity of a data set cannot be determined by size alone; simple data sets can be large and vice versa. Further, the number of boundary points in a data set can be expected to increase with the data's complexity. For this reason, the boundary point count was selected to represent complexity.

Figures 88 through 90 show how the RI phase scales for convergence window sizes of $q = 10$, $q = 100$, and $q = 1000$, respectively. Each figure contains two distinct trend lines. The first plots the execution time for the single processor configuration; the second reflects the best performance achieved by the multi-processor configurations tested. The trend lines are computed by fitting a polynomial (of degree 2) to the associated data for each type of processor configuration. All of the figures indicate a super-linear increase in execution time for the single processor configuration as the number of boundary points increases. For the multi-processor case, the trend line can be characterized as linear (or even sub-linear). Thus, it would appear that the RI phase algorithm scales far better in practice than predicted in Chapter V. It must be noted, however, that these findings are based on a very small group of data sets. Even so, it does provide some level of confidence that GRaCCE's run-time execution performance scales gracefully as the data increases in complexity.

With regard to the effect of size on scalability, the run-time execution times for the RI phase is measured for two data sets (Mushroom and SYN04) as the number of instances is increased. For the Mushroom data, a randomly selected subset of the data (corresponding to the indicated size) was extracted from the data set. In

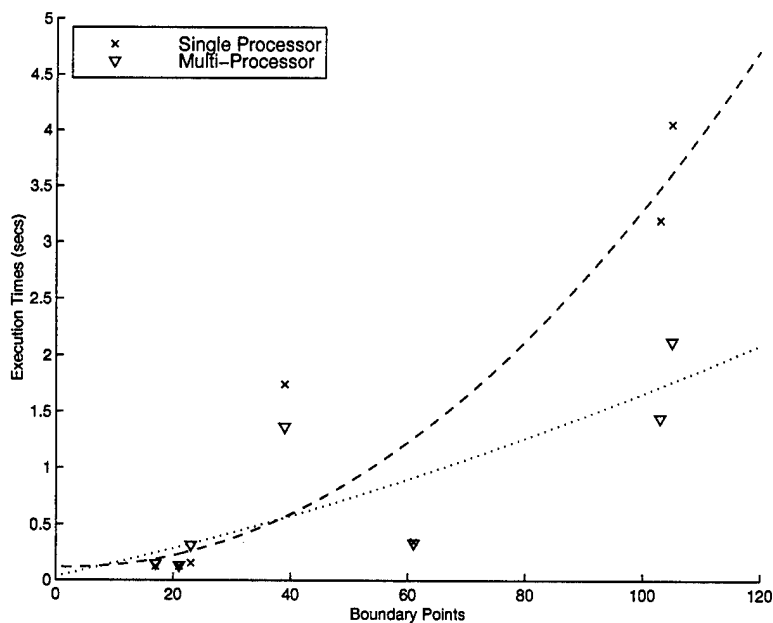


Figure 88 Scalability of Region Identification Phase for $q = 10$ as data set complexity increases.

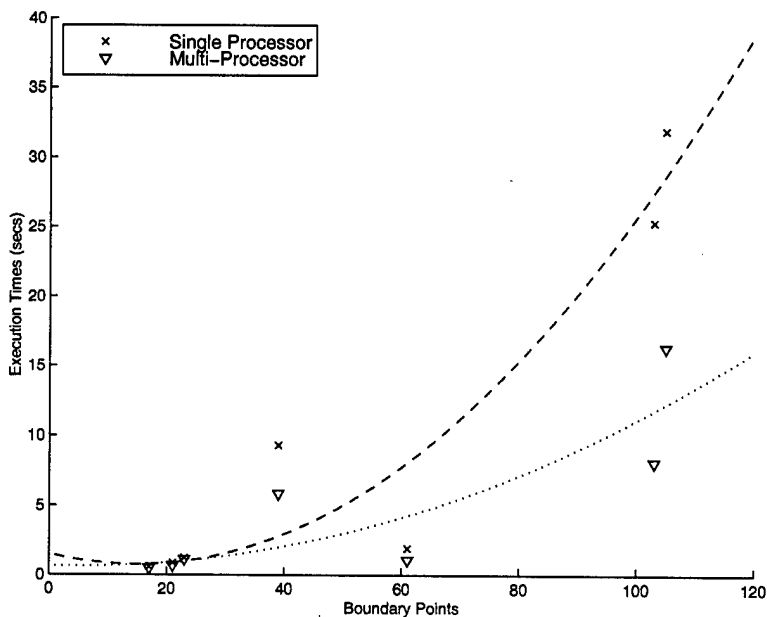


Figure 89 Scalability of Region Identification Phase for $q = 100$ as data set complexity increases.

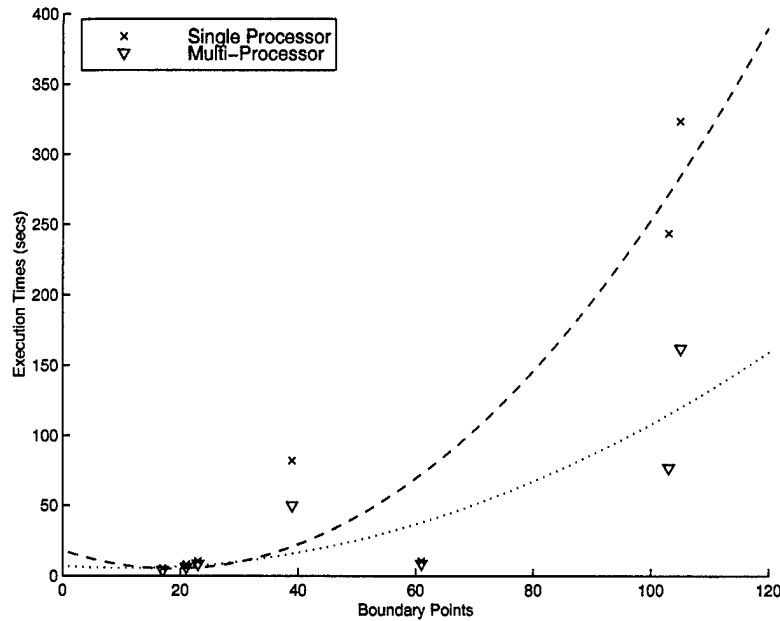


Figure 90 Scalability of Region Identification Phase for $q = 1000$ as data set complexity increases.

the case of SYN04, the specified number of instances were randomly generated from a predefined probability distribution. Figures 91 and 92 show the results for the Mushroom and SYN04 data sets, respectively. In both cases, the observed execution time levels off once the data reaches a critical size. In some respects, this outcome is intuitive since the boundary point set only reflects the probability distribution of the data. Once Λ is sufficient to approximate this distribution, the execution time of the RI phase should be fairly constant. These results are also consistent with the RI phase *average case* time complexity derivation in Chapter V, which show execution time growing as a function of the number of classes (m) and features (d), rather than data set size (n). It is important to note, however, that the growth rates of phases preceding the RI phase *are* directly dependent on n . The good news is that these phases contain operations (such as kNN) which can easily be parallelized (9); recall that Section 4.10 discusses the design issues involved with making GRaCCE concurrent.

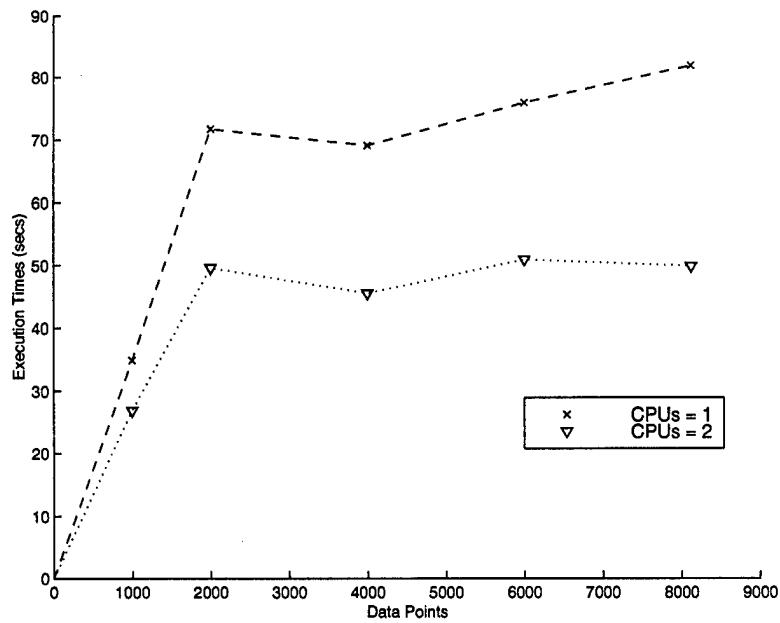


Figure 91 Scalability of cGRaCCE as the size of the Mushroom data set increases ($q = 1000$).

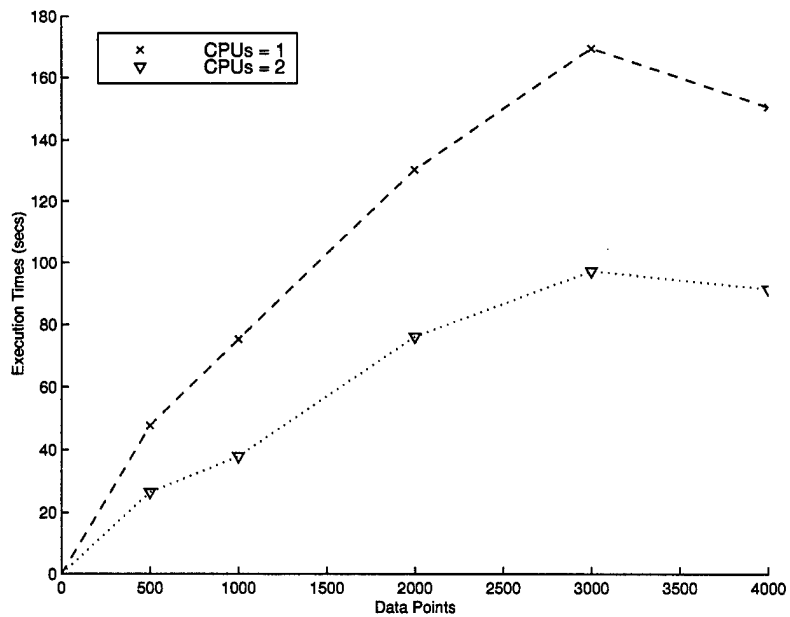


Figure 92 Scalability of cGRaCCE as the size of the SYN04 data set increases ($q = 100$).

6.8 Summary

The purpose of this chapter was to gauge how well GRaCCE performs in practice. In order to accomplish this, we tested the system on a variety of data sets (refer to Table 15). The types of experiments performed fell into two basic categories: inter-algorithm tests where GRaCCE was compared to other rule induction systems and intra-algorithm tests which evaluated the performance of different system modes. A detailed summary of conclusions drawn from the outcome of these experiments is provided below.

- The classification accuracy of GRaCCE is roughly equivalent to that of the decision tree algorithms tested. It under performs these algorithms, however, on those data sets primarily composed of discrete features.
- The rule sets produced by GRaCCE generally exhibit better generalization (in terms of classification accuracy) as compared to those of CART and C4.5.
- The classification error rate of GRaCCE generated decision rule sets can be incrementally reduced by using Mahalanobis distance to classify orphan data. However, this improvement is achieved by bypassing the generated rule set.
- GRaCCE consistently yields decision rules sets that are more compact (in terms of number of rules and conditions per rule) than CART, C4.5 and OC1.
- When rule set complexity is analyzed with respect to the number of terms in the rule set, GRaCCE clearly outperforms CART and OC1; it is comparable in performance to C4.5, however.
- The performance of GRaCCE on cross-validated data were more consistent than its competition. On some data sets (i.e., FLIR, glass and Soybean), the decision tree algorithms exhibited wide swings in terms of the measured results, depending on which portion of the data was used for training. This suggests that these methods require more training data than GRaCCE to yield consistent results.

- The quality of the generated partitions (and hence the rule set) may suffer when the training data is sparse with respect to the dimensionality of the data set. It must be noted, however, that this problem is not limited to GRaCCE (58).
- Testing revealed the system to be robust with respect to the minimum purity parameter (δ_{min}). Sensitivity to this parameter, however, is dependent on the characteristics of the underlying data set. In particular, data sets which are highly multi-modal are most impacted by variations in δ_{min} .
- The partition simplification process can significantly reduce the complexity of the decision rule set. However, for many rule sets this results in a small increase in the error rate. In addition, the worst case time complexity of this process, $O(d^4 n^2 \log(n))$, may make it prohibitive for large, high-dimensional data sets.
- While a concurrent version of GRaCCE was successfully demonstrated, its sub-linear speedup of execution time was less dramatic than predicted in Chapter V. This was due in large part to cache memory and interprocess communication overhead on short duration tasks. Given this, cGRaCCE should achieve higher speedup rates for more complex data sets (i.e., highly multi-modal, many boundary points).
- The scalability of cGRaCCE's execution time was tested for a limited number of data sets. The results indicates that, in practice, the multi-processor configuration scales in a linear manner as data set complexity increases. In addition, it appears that execution time of the RI phase may stabilize once a critical mass of boundary points is generated.

These findings indicate GRaCCE generated rule sets outperform those of state-of-the-art rule induction algorithms in terms of generalization and rule set complexity while maintaining a comparable level of accuracy. Because its decision rule sets are smaller, the individual rules have greater support than those of the competition. As a result, the knowledge discovered by the system is likely to be more valuable to the user. In addition, the benchmark results show GRaCCE can be parallelized and

demonstrates near linear scalability (in practice). Because these trends are based on the analysis of empirical results, there is no guarantee that they can be duplicated for every data set. That being so, the base of test cases is sufficiently diverse to reasonably conclude that they will apply to the vast majority of data sets for which the assumptions of Section 1.5 apply.

VII. Conclusions and Recommendations

This chapter presents a summary of the research discussed in this dissertation. The significant contributions of the research are outlined and recommendations for future research are provided.

7.1 Research Summary

The induction of decision rule sets from data that are accurate, compact and understandable is a fundamental goal of data mining. While Chapter II describes a wide spectrum of such decision rule induction (DRI) algorithms, many of these fall short of this ideal due to one or more of the following factors:

- The NP-complete nature of the problem.
- The greedy nature of the induction algorithm.
- Simplifying assumptions about the data.
- Simplifying assumptions regarding the rule structure (e.g., conditions are equivalent to univariate, linear partitions).
- Partitioning criteria that are inconsistent with the natural class boundaries.
- The tendency of induction algorithms to over-fit the data.

To illustrate this point, consider the traditional GA-based classifiers pioneered by Holland and Reitman (60) and DeJong et al (24). These methods use univariate class partitions to construct rules which are based on the categories of discrete features. A fundamental weakness of this approach is that these partitions may be arbitrary with respect to the clusters within the data; this is especially a concern when continuous data is categorized (33). The presence of such a condition can cause the evolved decision rule sets to have large error rates or be overly complex.

The above example highlights just how crucial a role the *inherent structure* of the data plays in the induction process. Many DRI methods are extremely time efficient due to a priori assumptions made about the structure of the data and the rule set. If the data fits these assumptions, then the results can be quite good; if not, then it becomes a case of trading execution speed for rule set quality (in terms of classification accuracy and size). A primer on data set related issues that impact DRI algorithms was provided in Chapter III.

The research discussed in this dissertation accomplished the primary goal of developing a new paradigm for a GA-based DRI, known as the Genetic Rule and Classifier Construction Environment (GRaCCE). This approach breaks with existing GA-based DRI methods by constructing a rule set from a pool of class partitions which are piecewise linear approximations to the Bayes optimal decision surface. As discussed in Chapters IV and V, it is possible to use combinations of these partitions to find class homogeneous (CH) regions in the data. Given a sufficient set of partitions, the GA serves as the search engine for this process. The chromosome structure enables each individual in the GA population to represent a CH region. Thus, GRaCCE harnesses the power of genetic search to find CH regions that meet a user-specified purity criteria while maximizing coverage and minimizing complexity (in terms of the number of partitions that define each region). Once located, these CH regions (and their associated partitions), can be simplified and converted into simple, accurate, and understandable classification rules.

Another important aspect of DRI algorithms is their scalability. Ideally, as the size of a data set increases, the time required to mine it should grow in a linear fashion. Some decision tree algorithms, such as ID3 (105) have been theoretically shown to meet this goal (144). Researchers have also devised parallel architectures for ID3 which exhibit linear speedup with respect to the number of utilized processors (132; 154). In light of these efforts, a concurrent version of the Region Identification (RI) phase (referred to as cGRaCCE) was prototyped. Since the RI phase actually per-

forms the CH region search, the motivation here is to determine if similar results can be achieved by parallelizing GRaCCE.

7.2 *Test Summary and Conclusions*

Chapter VI of the dissertation discusses the testing of the GRaCCE algorithm. The tests performed fall into two basic categories: inter-algorithm and intra-algorithm. The inter-algorithm testing compares the system's performance to that of several decision tree algorithms on a suite of benchmark data sets. The metrics of interest are classification accuracy and complexity of the generated decision rule sets. The intra-algorithm testing examines how different modes of GRaCCE affect performance. Specifically, the following issues are analyzed:

- What are the classification accuracy versus complexity reduction tradeoffs in invoking the partition simplification process?
- To what degree can accuracy be improved by classifying orphan data?
- How consistent are the results when the required CH region purity is varied?

Lastly, testing was conducted on the cGRaCCE prototype. These tests focused on the relative speedup (efficiency) of the system in a multi-processor configuration. In addition, the scalability of the RI phase was also evaluated in terms of increasing data set size and complexity. The essential conclusions regarding the strengths and weaknesses of GRaCCE (based on this testing) are¹:

- GRaCCE has classification accuracy comparable to those of the decision tree algorithms tested. It under performs these algorithms, however, on those data sets primarily composed of discrete features.
- The rule sets produced by GRaCCE exhibit better generalization (in terms of classification accuracy) as compared to those of CART and C4.5.

¹For a more detailed list of findings, refer to Section 6.8

- GRaCCE yielded decision rules sets that are significantly more compact (in terms of number of rules and conditions per rule) than any of the decision tree algorithms tested.
- The performance of GRaCCE on cross-validated data was more consistent than its competition. On some data sets, the decision tree algorithms exhibited wide swings in terms of the measured results, depending on which portion of the data was used for training. This suggests that these methods require more training data than GRaCCE to yield consistent results.
- While a concurrent version of GRaCCE was successfully demonstrated, for the most part it exhibited sub-linear execution time speedup. This was due in large part to cache memory and interprocess communication overhead on short duration tasks. Given this, cGRaCCE achieves best results for more complex data sets (i.e., highly multi-modal, many boundary points).
- In terms of scalability, results indicate that, in practice, GRaCCE's multi-processor configuration scales in a linear manner as data set complexity increases. In addition, it appears that execution time of the RI phase may stabilize once a critical mass of boundary points is generated.

The above results show that GRaCCE possesses a number of qualities which make it an excellent DRI algorithm. In particular, its ability to generate compact decision rule sets of equivalent accuracy (as compared to decision trees) is noteworthy. Additionally, it has an architecture that can be easily parallelized (refer to Section 4.10) and has shown interesting scalability properties for its most critical phase. Given these findings, GRaCCE has the potential to be a world class data mining paradigm.

7.3 Contributions

The principal contribution of this research is the development of GRaCCE as a general purpose method for the induction of simple and accurate decision rule sets

from data. The test results presented in Chapter VI show that, relative to several decision tree algorithms, GRaCCE evolves a more compact rule set while achieving a comparable level of accuracy for a wide range of data sets. This was accomplished by identifying characteristics essential to the success of other rule induction methods and combining them into a single, integrated algorithm; some of these characteristics include:

- Using estimates of natural class boundaries within the data set as the basis for constructing the decision rule set.
- Employing evolutionary search techniques to perform decision rule induction.
- Making a minimum of assumptions regarding the structure of the induced rules and/or the data.
- Removing noisy or easily misclassified instances from the training data set.
- Incorporating a method of simplifying the induced rules.
- Providing an architecture that allows rule induction to be accomplished with a high degree of parallelism and scalability. This is an essential characteristic for processing high dimensional data sets.

The following subsections summarize the secondary contributions resulting from this research. Each of these describe specific aspects of GRaCCE that serve to distinguish it from other rule induction algorithms.

7.3.1 A method for generating a sufficient set of partitions for decision rule construction. The key to GRaCCE's success is the ability to generate a *sufficient* pool of salient class partitions which can be used to form CH regions. As discussed in Chapter IV, this method has two primary components. The first is winnowing the data set in order to remove noisy and/or easily misclassified instances. This process makes the classes linearly separable. As a result, the identification of boundary point

pairs that straddle the Bayes decision boundary becomes much easier; this, in turn, lays the foundation for generating the initial set of class partitions.

The second component of the method uses the existing boundary point (Λ) and partition (Γ) sets to generate a sufficient partition set. The pseudocode describing this process is contained in Figure 29. It is proved in Chapter V that this process guarantees that Γ is sufficient to separate all boundary points of differing class present in the data set.

7.3.2 The development of a GA-based “0/1” approach for supervised clustering of data. The centerpiece of GRaCCE is the Region Identification phase, where CH regions are formed using the set of generated partitions. This is accomplished using a unique strategy where individual boundary points are made the focus of each search. Once a boundary point is selected, a GA chromosome is organized to enable CH regions to be represented as a fixed-length, binary string. The fitness of a region formed by such a chromosome is then evaluated on the basis of its coverage, purity and complexity. A full discussion of this approach can be found in Section 4.5.

7.3.3 A method for approximating the training set to accelerate rule induction. The use of weighted boundary points for rule induction gives GRaCCE two important advantages over other methods. First, it accelerates the search for CH regions since $|\Lambda|$ is typically a small fraction of the total data set size. Other DRI methods (such as decision trees) cannot substitute Λ for the data set because they do not generate partitions a priori. As a result, the *curse of dimensionality* (8) makes it infeasible for these algorithms to effectively utilize the boundary point set (by itself). Another advantage is that (in some cases), GRaCCE’s execution time stabilizes as the size of the data set increases; recall that this was observed with the Mushroom and SYN04 data sets in Section 6.7.2. This shows that using Λ in lieu of the data set itself can help to improve the system’s scalability.

7.3.4 The development of a concurrent architecture for GRaCCE. The design for a concurrent version of GRaCCE was introduced in Section 4.10 and subsequently implemented; the corresponding test results were presented in Section 6.7. Although sub-linear speedup was observed for small data sets due to cache memory and interprocessor communication overhead, analysis (and additional testing) indicate that the system's scalability should improve as the task duration increases sufficiently to compensate for these factors (49).

7.4 Recommendations for Future Research

It is relatively rare for research to have a finite beginning or ending. No matter how excellent the outcome of a research effort, there is always room for improvement and new approaches to be tried. With respect to this dissertation effort, the following subsections discuss ways (in order of priority) that the baseline GRaCCE system can be modified and/or extended through subsequent research investigations.

7.4.1 Translate GRaCCE from MatLab to C++. As indicated in Chapter VI, GRaCCE's run-time execution performance was not directly compared to those of the decision tree algorithms due to its implementation in *MatLab*. Although *MatLab* is an excellent environment for prototyping, its interpreted code is significantly slower than the executable of a compiled high level language (HOL) such as *C++*. In retrospect, coding GRaCCE in *C++* would have allowed for more extensive testing, due to a reduction in the time required to process a given data set. Given this, converting GRaCCE to *C++* should be the first order of business for any future research effort.

7.4.2 Develop a Fully Parallel Version of GRaCCE. It is widely recognized that parallel data mining algorithms are an effective tool for analyzing large databases (125). Unfortunately, only a single phase of GRaCCE was modified to execute concurrently. This is not because it is infeasible to parallelize the other

phases, but that the effort was focused on the most difficult part of the algorithm. As discussed in Section 4.10, it should be relatively easy to parallelize procedures like winnowing, boundary point identification and initial partition generation. In general, this can be accomplished by sharing Ω , Λ and Γ among all nodes in a multi-processor configuration and assigning a different, but equal, *slice* of the task to each node. Consequently, a prime goal for future research is to unlock the power of GRaCCE by developing a fully concurrent version.

7.4.3 Adapt for Very Large Databases. As discussed in Section 3.4, most real-world databases differ in terms of size and format from those tested in Chapter VI. While developing a fully parallel version of GRaCCE is an essential part of any approach for processing VLDBs, it is only one component. For example, a strategy must be developed for limiting the amount of main memory used by GRaCCE when processing VLDBs. While elements of the current approach (such as data set approximation and the use of a finite partition pool) support this objective, further innovations are needed. In addition, the larger issue of accomplishing automated mining of RDBs must also be addressed (although this issue is not specific to GRaCCE).

7.4.4 Optimize GRaCCE for Discrete Data Sets. One of the more important observations made in Chapter VI is that the decision rule set produced by GRaCCE are less accurate (relatively speaking) for data sets with discrete features than those composed of real-valued features². While it may seem that oblique partitions are not compatible with categorical data, it is important to remember that OC1 (which uses oblique partitions) (94) outperformed GRaCCE in these cases. Given this disparity, more research must be done in order to improve the quality of oblique partitions generated for discrete data sets.

²This statement is based on a performance comparison between GRaCCE, OC1 and C4.5

7.4.5 Application of Genetic Programming to GRaCCE. Because GRaCCE uses a binary GA chromosome, it is only possible to represent *convex* regions. While these types of regions are usually adequate, it may be possible to improve classification accuracy by using GPs (rather than GAs) to find regions of more complex shapes. Under this scheme, a separate GP would be evolved to represent each CH region. The GP terminal set could consist of the class partition set used by the GA. In addition, the GP function set would consist of logical operators (such as AND, OR, NOT, NOR, etc) which would serve to combine the partitions. The primary disadvantage of this approach is that the regions could be much harder to interpret than those evolved by the GA. Even so, this would be a worthwhile research topic.

7.5 The Last Word

In conclusion, the design, development and testing of GRaCCE have provided a tremendous amount of insight into a number of disciplines, including data mining, machine learning, and pattern recognition. Given the many ways this technology can be applied to the USAF mission, it is my sincerest hope the metaphoric gauntlet will be picked up by subsequent AFIT students interested in data mining research.

Appendix A. Review of Genetic Search Methods

This appendix provides a short tutorial on the genetic algorithm (GA) and genetic programming (GP) paradigms¹

A.1 What is a Genetic Algorithm?

The *genetic algorithm* is a stochastic global search method which derives its behavior from a metaphor of some of the mechanisms of *evolution* in nature. This is done by the creation within a machine of a *population of individuals* represented by *chromosomes*, in essence a set of character strings that are analogous to the base-4 chromosomes that we see in our own DNA. The individuals in the population then go through a process of simulated evolution in a given environment. In this case, the environment can be thought of as a fitness landscape (70) like that shown in Figure 93; each member of the GA population occupies a point on the landscape. Just as in natural adaption, the evolutionary process results in a population of individuals better suited to their environment than the initial population.

In practice, this genetic model of computation can be implemented using arrays of bits or characters to represent the chromosomes. Simple bit manipulation operations allow the implementation of *crossover*, *mutation* and other operations. Although a substantial amount of research has been performed on variable-length strings and other structures, the majority of work with GAs (including the research described in this document) is based on using fixed-length character strings to encode the solution being sought. This is a crucial characteristic that distinguishes genetic algorithms from *genetic programming*, which does not have a fixed length representation and there is typically no encoding of the problem.

¹Much of the material in this appendix is taken from existing tutorials on evolutionary search methods (16; 46; 56; 92).

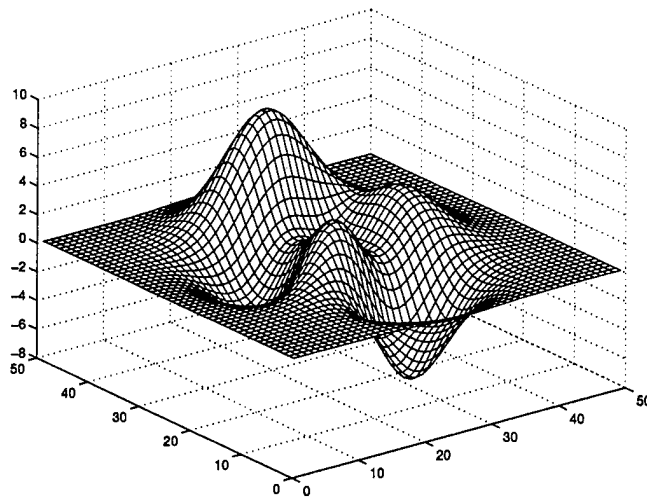


Figure 93 The GA is used to search a given landscape.

A.2 Algorithm

When the genetic algorithm is implemented, it is usually done in a manner that involves the following cycle: Evaluate the *fitness* of all of the individuals in the population. Create a new population by performing operations such as crossover, fitness-proportionate *reproduction* and mutation on the individuals whose fitness has just been measured. Discard the old population and iterate using the new population.

One iteration of this loop is referred to as a *generation*. There is no theoretical reason for this as an implementation model. Indeed, we do not see this punctuated behavior in populations in nature as a whole, but it is a convenient implementation model.

The first generation (generation 0) of this process operates on a population of randomly generated individuals. From there on, the genetic operations, in concert with the fitness measure, operate to improve the population. This cycle continues for a maximum number of generations (specified by the user). The basic pseudocode for a GA is presented in Figure 94. A more formal, mathematical GA definition is provided by Bäck (6).

```

Algorithm GA is
  // start with an initial time
  t := 0;

  // initialize a usually random population of individuals
  initpopulation P(t);

  // evaluate fitness of all initial individuals of population
  evaluate P(t);

  // test for termination criterion
  while t < max_generations (q)

    // increase the time counter
    t := t + 1;

    // select a sub-population for offspring production
    P' := selectparents P(t);

    // recombine the "genes" of selected parents
    recombine P'(t);

    // perturb the mated population stochastically
    mutate P'(t);

    // evaluate its new fitness
    evaluate P'(t);

    // select the survivors from actual fitness
    P := survive P, P'(t);

  end; // while
end GA.

```

Figure 94 GA Algorithm Outline.

The GA is of polynomial order complexity with a finite space requirement determined by the population size. Although Goldberg (42) suggests that there is an optimal population size (n) which depends on the length of the chromosome (l), there is no fixed dependence between the length of the string and the population size. Experimental evidence, however, suggests that an insufficient population size may adversely affect solution quality (41; 92).

The various genetic operators have associated maximum complexities, although the complexity of an actual implementation may differ. The complexity of each operator is discussed in Section A.5. The complexity for the entire algorithm, however, has been derived as

$$O(q \times n \times \max(l, \Phi, n)) \quad (48)$$

where q is the maximum number of generations, n is the number of individuals in P , and Φ is the GA objective function (78).

A.3 Theory

Since the theory of evolution is predicated on survival of the fittest, individuals that exceed the population's mean fitness level are more likely to pass on their genes. This principle is the basis of Holland's Schema Theorem (61) which is the cornerstone of GA theory. Due to its importance, this section reviews the basic theory underlying the Schema Theorem.

A.3.1 Schema. Goldberg develops an estimate for the performance of the Simple GA (SGA) (41). Theoretical analysis of GA performance makes extensive use of *schemata*, or similarity templates. Schemata are strings composed of characters taken from the genetic alphabet, with the addition of the "don't care" character (*).

A schema thereby describes a subset of the potential solutions. For example, the schema 1***** represents the set of all 8-bit strings which contain a 1 in the first position. Likewise, the schema 1*****0 represents the set of all 8-bit strings which begin with a 1 and end with a 0.

The defining length, $\delta(H)$, of a schema is the “distance” between the index of the first specified position and the index of the last specified position. For example, $\delta(1*****0*) = 7 - 1 = 6$, while $\delta(1******) = 1 - 1 = 0$. The order of a schema H , which is denoted $o(H)$, is the number of specified positions in the schema. For example, $o(1******) = 1$, while $o(11111111) = 8$.

The schema concept can be extended to apply to absolute and relative ordering problems. Following Kargupta (69), an absolute ordering schema defines a set of valid permutation strings. For example, the absolute o-schema !1!5!! represents the set of all permutation strings for which the second and fourth positions contain alleles 1 and 5, respectively. This o-schema is distinct from the standard schemata *1*5** in that the former requires that the string represent a valid permutation, while the latter does not.

A.3.2 Fundamental Theorem. Defining the average fitness of a string matching a schema H to be $f(H)$, the average population fitness to be \bar{f} , and the number of strings in a population at time t which match the schema to be $m(H, t)$, the effect of the reproduction operator is

$$m(H, t + 1) = m(H, t) \frac{f(H)}{\bar{f}} \quad (49)$$

Noting that crossover disrupts a schema only when the crossover point occurs within the defining length of the schema, the probability of survival under crossover for a schema in a string of length l is

$$p_s \geq 1 - p_c \frac{\delta(H)}{l-1} \quad (50)$$

where p_c is the probability of crossover and the inequality is used to reflect the fact that crossover may not actually disrupt the schema even when the crossover point is within the defining length.

The probability of survival for the above schema under the mutation operator then can be estimated as

$$p_{ms} \approx 1 - o(H)p_m, p_m \ll 1 \quad (51)$$

where p_m is the probability of mutation. Combining these results and omitting the negligible terms gives an estimate for the expected number of examples of a schema in the next generation

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m \right] \quad (52)$$

This is referred to as the Fundamental Theorem of Genetic Algorithms, and can be interpreted by stating that “short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations” (61). This result also goes by the name of the Schema Theorem.

A.4 Problem Encoding

In the GA paradigm, individuals are coded as strings (chromosomes). Within the chromosome are contiguous string segments (genes) that represent distinct fea-

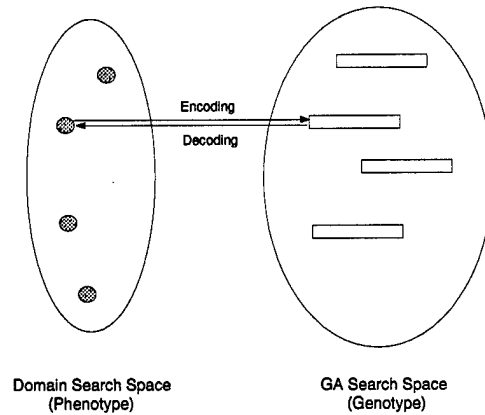


Figure 95 The objective function translates an encoded genotype to its fitness as a solution in the original domain search space.

tures. The encoding must be such that the objective function can uniquely map the chromosome's value (genotype) to a corresponding fitness value (phenotype); Figure 95 gives a pictorial representation of this translation process. GA designers have traditionally used binary strings to encode the chromosome. One reason for the popularity of binary strings is that they offer the maximum number of schemata (for a string of fixed length) of any discrete code (6). The practical advantage of having more schemata is that it provides more flexibility in tuning the solution. Binary strings are also convenient to use because they are easily manipulated by GA operators such as crossover and mutation. They can also be used to represent non-binary numbers that have integer and floating point representations.

Genetic algorithms are used for a number of different types of problems. One category is *functional optimization* problems. In these cases, the GA searches for the best set of input parameters to optimize a given function. The parameters are typically encoded by genes which use a set of contiguous bits to represent the required range of values. An example coding of a 4 variable parameter set with a binary GA is shown in Figure 96. The objective function decodes these parameter value and inputs it to the function being optimized.

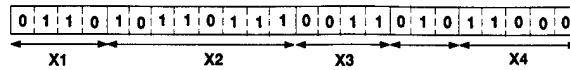


Figure 96 Binary GA Chromosome Format (4 Variable).

In the context of this dissertation, the GA is used to solve a “0/1” type problem, where each individual represents a discrete set of items included in the solution set. With this encoding, each item is uniquely represented by a single gene. Thus, when a gene’s allele is set to 1, the item is present in the solution set; the opposite is true when the allele is set to 0. The objective function then evaluates the goodness of the given solution. As a result, these types of problems fall into the combinatorial optimization category since the objective is to search for the optimal solution set. The Knapsack problem (95) is an example of a 0/1 type problem to which GAs are frequently applied.

Given the discussion of GA theory provided above, it is desirable to code the chromosome such that schema contributing to good solution fitness have a short defining length (δ). This lessens the chance that such schema will be disrupted by either crossover or mutation operators. In many problems, however, the inherent epistatic relationships between genes are not known by the GA designer. Methods have been developed, such as Goldberg’s Messy GA (44), to find good schema and organize the chromosome accordingly.

A.5 *Reproduction Strategies*

Reproduction is perhaps the most important process within a GA because it determines the makeup of the next generation. Consequently, the reproduction strategy controls the direction of the search. As indicated in Figure 94, the reproductive process can be decomposed into a number of sub-functions: selection, recombination, mutation, evaluation, and reconstitution (of the population). Each of these is summarized in the subsections that follow.

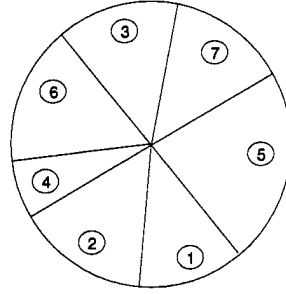


Figure 97 Roulette Wheel Selection.

A.5.1 Selection. Selection is the process of determining the number of times, or trials, a particular individual is chosen for reproduction; in turn, this influences the number of offspring a given individual will produce. Most selection scheme are based on fitness to insure that the most deserving individual are chosen to pass on their genes (to enforce the Schema Theorem). In addition, they are also probabilistic in nature. This helps insure diversity of the population, thus avoiding premature convergence.

The two types of selection used in GRaCCE are roulette wheel selection (RWS) and stochastic universal sampling (SUS). In RWS, a *sum* is computed from the sum of the expected selection probability of all individuals, based on fitness. Each individual is then mapped to the contiguous interval $[0, sum]$; the size of the interval is determined by the individual's relative fitness. Consider, the roulette wheel depicted in Figure 97. Since individual 5 is the most fit, it occupies the largest interval on the wheel; in contrast, individual 4 is the least fit and has the smallest interval. To select an individual, a random number is generated in the interval $[0, sum]$; the individual whose segment spans the this number is chosen. This process is repeated until the required number of individuals is selected.

Instead of the single selection pointer employed in RWS, SUS uses n equally spaced pointers, where n is the number of selections required. The population is randomly shuffled and a single random number (*ptr*) in the range $[0, sum/n]$ is generated. The n individuals are then chosen by generating a pointer for each individual,

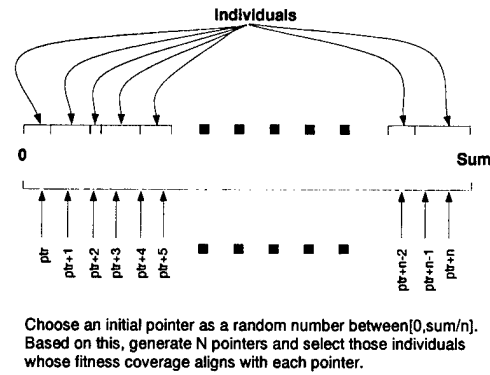


Figure 98 Stochastic Uniform Selection.

based on ptr , spaced by 1. For example, the pointer for the first individual is ptr ; for the second, it is $ptr+1$, for the third, it is $ptr+2$, etc. The individual which spans each of the n pointer positions is selected. Figure 98 provides an illustration of this process. Because it relies on only a single random number, SUS has a complexity of $O(n)$; this makes it much simpler than RWS, which has a complexity of $O(n \log n)$

A.5.2 Recombination. The basic operator for producing new chromosomes in the GA is that of crossover. Like its counterpart in nature, crossover produces new individuals that have parts of both parent's genetic material. This operation serves as the means to redistribute the higher fitness building blocks of a population as it forms new candidates from the current population. The recombination algorithm takes the individuals selected for reproduction and puts them in an array. Since the procedure requires two parents, individuals with even indices are always mated to their adjacent counterparts (odd) with the specified probability of crossover. Once sets of parents are chosen, a crossover operator is applied. The three types of crossover operators used in GRaCCE are single point, double point and uniform; each of these are briefly described below. Each of these can be considered to be of $O(l)$ since they traverse the length of the string.

A.5.2.1 Single Point Crossover. The single point crossover operator randomly picks a crossover position in the chromosome. As Figure 99 depicts, the

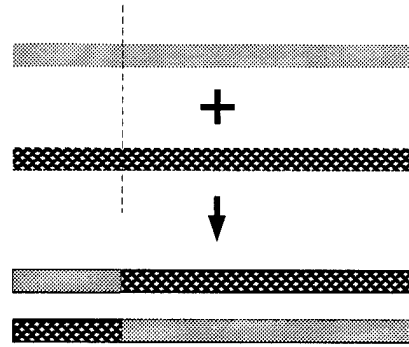


Figure 99 Example of single point crossover.

two children are composed of material from each parent (from either side of the chosen position). Because the single crossover point is chosen by a uniformly random function, high order schema are most likely to be disrupted. As a result, single point crossover is said to be biased toward short, low order schema.

A.5.2.2 Double Point Crossover. As the name implies, double point crossover picks two crossover points. This results in children that have alternating segments from both parents (see Figure 100). Multi-point (3+) operators accomplish similar matings, albeit with more crossover points. While these operators help eliminate some of the bias toward high order schema (since only a section of the schema may be changed), it is still highly disruptive; this is not necessarily a bad thing, however. It has been noted that the disruptive nature of multi-point crossover appears to encourage the exploration of the search space, rather than favoring the convergence to highly fit individuals, making the search more robust (128).

A.5.2.3 Shuffle Crossover. Even though double point crossover reduces some of the positional bias associated with single point crossover, it does not eliminate it. For this to occur, a random, bitwise crossover operator (also known as uniform) is needed. While the *MatLab* GA Toolbox does not offer this, it does have a shuffle crossover operator. This operator works by first randomly shuffling the bits in both parents. It then produces offspring using a single point crossover. At this

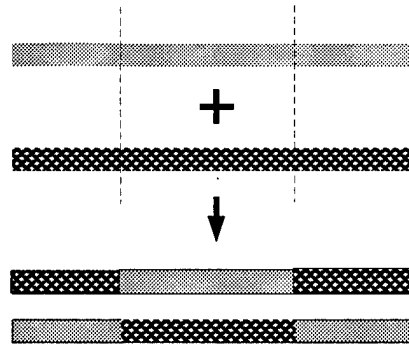


Figure 100 Example of double point crossover.

point, it then returns the bits in the offspring to their original position (as found in the parent). An illustration of this process is contained in Figure 101.

A.5.3 Mutation. In natural evolution, mutation is a random process where one allele of a gene is replaced by another to yield a new genetic structure. In GAs, mutation is randomly applied, typically with low probability (in the 0.001 to 0.01 range) and modifies elements in the chromosome. For example, a mutation rate of 0.01 means that 1 bit in 100 will be changed. For a binary chromosome, mutation occurs by toggling the allele's value. Usually considered as a background operator, mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero. This operator also functions as a safety net to recover good genetic material that may be lost through the selection or crossover processes. The mutation operation can be up to $O(n)$ complexity.

A.5.4 Evaluation. This phase consists of the evaluation of each member of the population by an objective function. The objective function returns a value associated with each encoded solution that represents its goodness in the context of the phenotype domain (refer to Figure 95). Often, some type of scaling function is employed to translate the raw fitness value that can be more easily processed by the other reproductive functions. For example, the raw fitness value yielded for minimization problems is not compatible with the SUS selection operator. As

P1	=	0	0	1	1	0	1	0	0	1	1
P2	=	1	1	0	1	1	0	0	1	1	0
MAP	=	8	4	7	9	3	2	5	1	0	6
SP1	=	1	0	0		1	1	1	0	0	0
SP2	=	1	1	1		0	1	0	1	1	0
^ - Crossover Point											
S01	=	1	1	1		0	1	0	1	1	0
S02	=	1	0	0		1	1	1	0	0	0
O1	=	1	1	1	0	0	1	0	1	1	1
O2	=	0	1	0	0	1	0	1	0	1	0

Figure 101 Example of Shuffle Crossover.

indicated by Equation 48, the objective function is a major driver of the GA's growth rate.

A.5.5 Reconstitution. The reconstitution operation builds a new population (the next generation) from members of the existing population (parents) and their offspring. In the current GRaCCE implementation, the existing population is completely replaced. However, the GA Toolbox gives the option of replacing the existing population with a fraction of the new. A uniform (random) or fitness-based replacement strategy is offered to choose which members of the existing population are replaced.

There are two types of GA reproduction: sexual and asexual. Sexual reproduction is accomplished by combining components of selected individuals together (parents) in order to form a new individual (child). Asexual reproduction involves the random modification (mutation) of an existing individual. Mutation helps the GA break out of local minima, thus aiding the search for a globally optimum solu-

tion. While there is no guarantee that GAs will find the best solution, their method of selection and breeding candidate solutions can cultivate a pool of “good” solutions given enough generations.

A.6 What is Genetic Programming?

Genetic Programming is a method of automatically generating computer programs to perform specified tasks (77). The GA and GP paradigms share many similarities. Like GAs, GPs emulate biological evolution in order to search through a landscape for an optimal solution. In addition, the basic process by which GPs are evolved is identical to that of GAs (see Figure 94). Despite the fact that they share similar fundamentals, GAs and GPs differ in several important ways. The first major difference concerns the structure used to represent an individual. In a GA, each solution is a chromosome of fixed length conforming to a predefined schema. In contrast, the solutions generated by GPs are executable computer programs. A GP solution candidate is composed of a set of functions and terminals. Functions are operators (such as $+$, $-$, $*$, and $/$) that accept parameters, process them, and return a value; functions can also accept the output of other functions as parameters. Terminals are variables that are passed to functions as parameters. The function and terminal sets utilized in a given GP are specified by the user.

An example of a GP which uses the above operator set is shown in Figure 102. As illustrated by the figure, GPs have a tree-like structure that can be readily translated into Lisp-like code. Note that the tree is evaluated recursively starting with the root node. The value returned by the root node is the output of the GP. Starting with the root node, the parameter of each function corresponds to a branch in the tree; each parameter can be satisfied by either another function (creating additional branches) or a terminal (leaf). In order for this scheme to work, any function must be able to accept as a parameter any other function or terminal; this condition is known as the closure property.

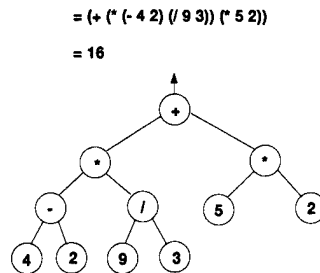


Figure 102 Example Genetic Program Structure.

The operation of the objective function also distinguishes the two paradigms. With a GA, the programmer mandates a method to solve the problem. Each GA solution candidate is processed by this algorithm; the objective function then evaluates the corresponding results. Since the algorithm is predefined (by the programmer), the GA can merely adjust variables (in the chromosome) to tune the algorithm's performance. Contrast this with the GP paradigm where the programmer provides the basic "building blocks" (i.e., function and terminal sets) to be used in constructing a solution. It is therefore imperative that the function/terminal sets provided are sufficient to solve the problem. Each GP solution is executed and the corresponding results are evaluated by the objective functions. In essence, the GP paradigm determines how best to construct a solution given the appropriate function and terminal sets. This gives GPs more flexibility in finding an optimal solution because the programmer tells it what to use, but not how to use it.

The final difference results from the way reproduction occurs under the two paradigms. While both GAs and GPs use the same operators (the crossover operator for sexual reproduction and the mutation operator for asexual reproduction), the manner in which they are applied is dependent upon each paradigm's solution representation. In the GA paradigm, crossover is performed by splitting two parent chromosomes at a randomly selected point (called the crossover point). The child is formed by combining a chromosome components from each parent (taken from opposite sides of the crossover point). In contrast, the GP paradigm accomplishes

sexual reproduction by swapping subtrees of the parents in order to form a child. Asexual reproduction is performed in the GA paradigm by randomly toggling bits in the parent chromosome; as a result, the child is a mutation of the parent. In the GP paradigm, mutation is accomplished by randomly picking a subtree in the parent and generating a new subtree to replace it.

Appendix B. GRaCCE User's Guide

The purpose of this appendix is to provide a rudimentary level of guidance for using the Genetic Rule and Classifier Construction Environment (GRaCCE). Since a detailed description of the GRaCCE algorithm was given in Chapter IV, the material presented here primarily serves to complement that discussion. Each of the following sections provides explanations of how to configure and invoke all available system operations. In addition, a series of useful hints for achieving optimal results are furnished. It is strongly suggested that this appendix be read in full before attempting to use GRaCCE.

B.1 Running GRaCCE

In order to use GRaCCE, the *MatLab* program must first be executed. This can be done by typing *MatLab* at the *Unix* prompt. Once *MatLab* is fully elaborated, the following sequence must be typed at the *MatLab* prompt to initialize GRaCCE:

```
>> addpath .../ga2  
>> addpath .../gracce  
>> gracce
```

The first two statements define the directory paths that GRaCCE needs to access as it executes. The first path specifies the location of the University of Sheffield Genetic Algorithm Toolbox software (16); this is the GA package used by GRaCCE. The second path point to the directory where the GRaCCE *MatLab* code resides. The third statement executes the program.

When GRaCCE elaborates, the Main menu (shown in Figure 103) will appear. All system functionality is accessible through the Main menu. Table 26 describes

the selectable items on this menu. Follow the instructions in this table to invoke any GRaCCE operation (select features, winnow data, rule induction) and select the related sub-menus. Note that when an operation is started, the menus will not accept any input until it completes.

B.2 Preprocessing Operations

Before a decision rule set can be generated from a particular data set, it is first necessary to pre-process it. As explained in Chapter IV, GRaCCE has two preprocessing operations: feature selection and winnowing; of these, only winnowing is mandatory.

B.2.1 Loading the Data File. The loading of a data set (in the correct format) is a necessary prerequisite for performing preprocessing. A data file must have a “*.dat” extension in order to be recognized by the system. The data files must be a flat file format, with a single feature vector specified per line. On each line, distinct elements must be separated by at least one space character; in addition, all elements must be in numeric form. The first element (leftmost) of each feature vector must be the class; unlike the other elements, the class must be in natural integer form (0, 1, 2, ...). Subsequent elements specify the values of the individual attributes for that feature vector; these may be in either integer or real form. A sample data file (for the Iris data set) can be found in Appendix C.

Once an acceptable data file is obtained, it can be loaded using the **Load Data Set** button. When pressed, this button brings up a browse menu to select the “*.dat” file. When the desired file is located and selected, pressing the **Done** button on the browse menu causes it to be input to GRaCCE. If it is successfully read in, the data set information fields described in Table 25 will be instantiated (they are initially blank). Before proceeding, check these fields to make sure they reflect the true content of the data set.

B.2.2 Feature Selection. If a reduced feature set is desired, then feature selection (FS) must be done prior to winnowing. The FS operation can be configured using the Preprocessing menu shown in Figure 27. FS can only be initiated after a data set is successfully loaded. When the operation finishes, the best feature set found is output to the “*.fsr” file. This file consists of a string of 1’s and 0’s, corresponding to selected and ignored features, respectively. If another (non-GRaCCE) method is used for FS, then an “*.fsr” file can be manually constructed.

B.2.3 Winnowing. Like FS, winnowing can only be initiated after a data set has been successfully loaded. Winnowing can either be performed on a full or reduced feature set, as specified in the *Feature Set Type* item in the Preprocessing menu. If a reduced feature set is chosen, then an “*.fsr” (matching the name of the “*.dat” file) must be provided. When the operation finishes, a “*.wds” file containing the winnowed data is generated.

B.3 Decision Rule Induction

The decision rule induction operation can be configured using the Genetic Algorithm (Figure 105) and Region Identification Phase (Figure 106) menus¹. Once a given menu is selected, it can be displayed by pressing the **Edit Menu** button. The items in each menu, and how they affect the algorithm’s execution, are described in Tables 28 and 29. The way in which the Table 28 menu parameters control the GA are discussed in Appendix A; likewise, Chapter IV explains how GRaCCE uses the Tables 29 menu items.

Before the induction process can commence, the appropriate “*.wds” file must be loaded². When this has been accomplished, the process can be initiated by pressing the **Execute Operation** button. As the algorithm executes, messages

¹Default item values are depicted in the referenced figures

²The browse menu filter is driven by the type of operation selected. Preprocessing operations use the “*.dat” filter. This procedure uses the “*.wds” filter.

indicating its progress are output to the *MatLab* window. When the operation has completed, all relevant results are written to a report file “*.rpt”; a sample report file can be found in Appendix C.

B.4 Helpful Hints

The way in which GRaCCE processes a given data set is driven by the data’s structural complexity. For example, highly multi-modal data sets are much more difficult to process than less complicated, ones. A data set’s complexity is dependent on the following factors:

- Data set size
- Data set dimensionality
- Number of classes
- Number of modes per class
- Relative position of these modes with respect to each other. Specifically, are modes of a given class clustered together or are they interleaved with those of other classes?

Of these factors, the first two are known a priori, while the last two are typically unknown (especially for high dimensional data sets). It is reasonable, however, to use the size of the boundary point and partition sets generated by GRaCCE as an estimate of the data set’s complexity. In general, the larger these sets, the more complex the data set. Knowing the relative complexity of a data set can be useful in choosing good parameter values. For this reason, it is good practice to perform a trial run on each new data set in order to determine the size of these items³. With a data set’s complexity characterized, the following guidelines can be applied:

³The **Simplify Partition On:** item should be set to NONE for trial runs

1. For most data sets (especially those that with many classes) it is a good idea to use the global-only partition set before using the mixed option (global and local partitions). This option will accelerate the algorithm's execution and provide a baseline error rate (for comparison purposes).
2. In order to remain competitive (in terms of execution time) with other rule induction algorithms, GRaCCE is typically run with small populations (100 members or less) and a small convergence window ($q = 10$).
3. As complexity increases, the percentage of utilized partitions can be decreased. This is because clusters in a fragmented data set tend to be most affected by the partitions closest to them.
4. The δ_{min} parameter should be set to a minimum of 70%. For highly multimodal data sets, purity should be set at a high value (90+%) to preserve the natural mode structure.
5. Increasing the size of k during the winnowing process generally reduces the size of the partition set generated. However, large values of k may cause small classes to be completely eradicated.
6. Choosing to simplify partitions using the full or winnowed training set will result in a compact decision rule set with the best accuracy. However, using these options can be prohibitive (in terms of processing time), especially for large, high dimensional data sets.
7. Setting the number of points for covariance matrix (Σ) estimation lower than the dimensionality can produce a singular Σ . Requiring too many points, however, can result in partitions that are unrepresentative of a given region. Thus, for data sets with high dimensionality (10^+), using a setting other than automatic may be a necessity.
8. The default **Probability of Mutation** (p_m) setting (0.1) is higher than usual. Traditionally, mutation rates fall in the 0.001 to 0.01 range, depending on the

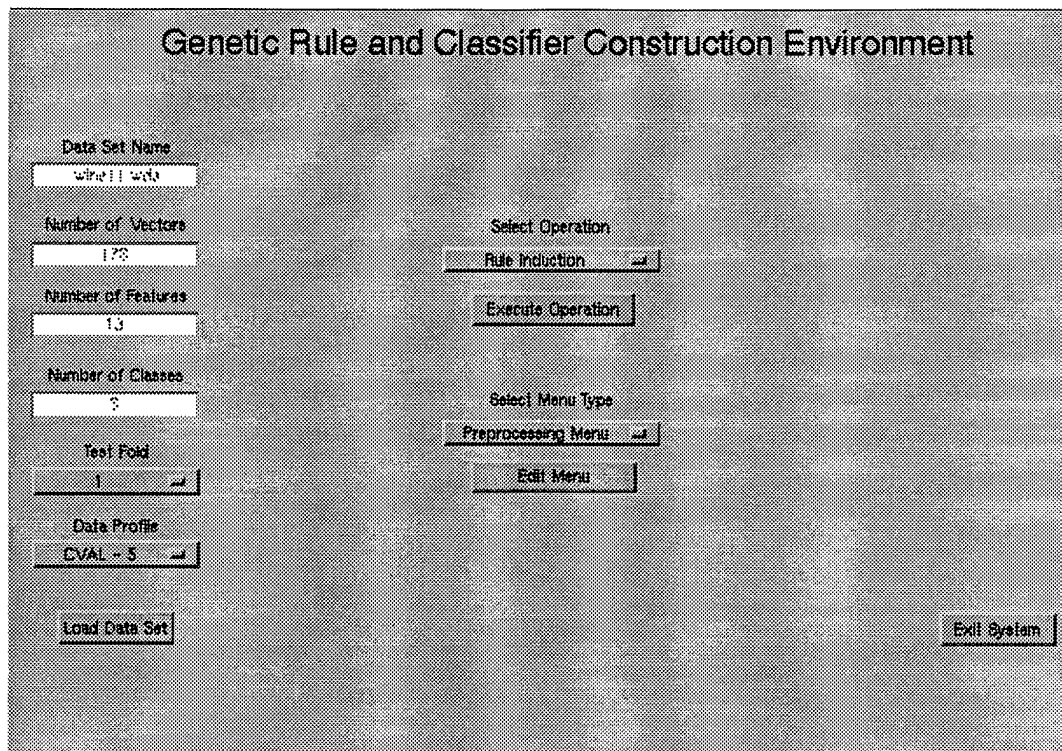


Figure 103 GRaCCE - Main Menu.

size of the GA chromosome (6). It was chosen, however, because it appeared to produce the best results for the data sets tested. This observation is supported by Tate and Smith (136), who found that high mutation rates are beneficial in combinatorial optimization problems. In addition, Spears and DeJong (128) showed that high mutation rates can help small populations avoid premature convergence.

B.5 Leaving GRaCCE

In order to terminate GRaCCE, press the **Exit System** button on the Main menu. If, however, GRaCCE is in the middle of an operation, processing must first be halted using the <CNTRL-C> key at the *MatLab* prompt.

Table 25 Main Menu - Data Set Informational Fields.

Field	Description
Data Set Name	Name of the loaded data set.
Number of Vectors	Total feature vectors in the loaded data set.
Number of Features	Total features in the loaded data set.
Number of Classes	Total classes in the loaded data set.

Table 26 Main Menu - Selectable Items.

Item	Description
Data Profile	This item determines how the data set is divided for training and test purposes. The choices are: 50/50, 60/40, 75/25, 80/20, 90/10, CVAL-5 and CVAL-10. For example, if 75/25 is selected, then a random sample of 75% of the data is randomly chosen for training and the remaining 20% is used for testing. If a cross-validation (CVAL) option is chosen, then the fold specified in the Test Fold item is selected for testing and the rest are used for training. For this option, the data set can be divided into either 5 or 10 folds.
Test Fold	If one of the CVAL options is selected in the Data Profile item, then this item chooses the data set fold used for testing. If a CVAL option is not selected, then this item is not displayed in the menu.
Select Operation	This item chooses the GRaCCE operation to be executed. The possible operations are select features, winnow data and rule induction. The first two are preprocessing operations; these require a virgin data set (*.dat) to be loaded before they can be executed. The rule induction operation produces the decision rule set; a winnowed data set (*.wds) must first be loaded. Data sets can be loaded by pressing the Load Data Set button. The selected operation can be run by pressing the Execute Operation button.
Select Menu Type	A particular parameter menu can be chosen with this item. The three available are the Preprocessing, Genetic Algorithm and Region Identification Phase menus. The selected menu can be edited by pressing the Edit Menu button. These menus are described in Tables 27 through 29.

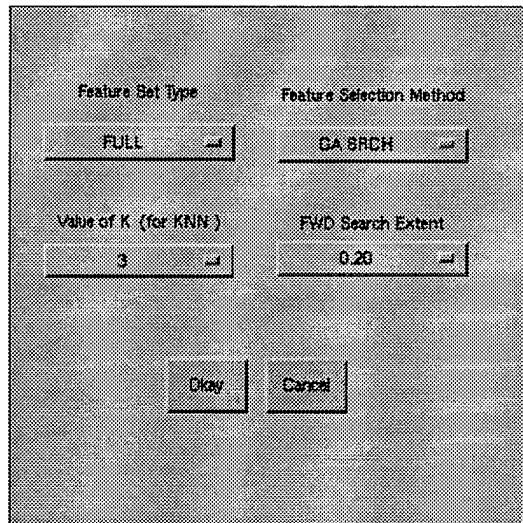


Figure 104 GRaCCE - Preprocessing Menu.

Table 27 Preprocessing Menu - Selectable Items.

Item	Description
Feature Set Type	Determines if full or partial feature set should be used during the winnowing process. If the partial set is chosen, the “*.fsr” file is read in to determine the correct features to utilize.
Feature Selection Method	Selects the method to use when feature selection procedure is executed. The available methods include: forward search (FWD), GA-based search (GA), or a hybrid of the two (FWD-GA).
Value of K	Determines the value of k for the kNN algorithm used in the preprocessing operations. The choices for k are: 1, 2, 3, 4, 5, 10, and 20.
FWD Search Extent	Specifies the percentage of total features to search for when a FWD or FWD-GA feature search is initiated.

Table 28 Genetic Algorithm Menu - Selectable Items.

Item	Description
Population Size (n)	Determines size of the GA population. The available sizes are 50, 75, 100, 200, 300, 500, 750, or 1000.
Probability of Crossover (p_c)	Specifies the crossover probability for mating individuals selected for reproduction.
Probability of Mutation (p_m)	Specifies the mutation probability for GA reproduction. This parameter ranges from 0.001 to 0.1. If the automatic option is chosen, the mutation rate is computed using an equation derived by Schaffer (112), $p_m = 1.75/(n\sqrt{l})$
Replacement Percentage	Determine fraction of the population to be reproduced. Values range from 0.5 to 3.0. A number greater than 1 allows more offspring to be reproduced than parents. A number less than 1 denotes an elitist selection strategy.
Selection Function	Type of probabilistic function used to select individuals for reproduction. Either stochastic universal sampling (sus) or roulette wheel sampling (rws) can be chosen.
Recombination Function	Determines how individuals are mated during reproduction. The available functions are: single point (xovsp), double point (xovdp), or shuffle (xovsh).
Convergence Window Size (q)	Number of generation that must pass without a change in fitness before the GA will terminate. The available window sizes range from 3 to 50 generations.

Table 29 Region Identification Phase Menu - Selectable Items.

Item	Description
Partition Utilization	Proportion of applicable partitions (closest to the chosen boundary point) utilized for CH region searches (default = 1.0).
Sample Size for Covariance Estimate	Selects the number of boundary point neighbors (of like class) used to compute the class covariance matrix during partition generation. The choices are: Automatic, 3, 5, 10, and 20. If the Automatic option is selected, the sample size is set equal to the number of features.
Region Utility Ratio	This value is the RUR threshold used during the region refinement phase to eliminate CH regions of poor quality. The available value vary between 0.001 and 0.3, with a default of 0.1. The larger the number selected, the more regions will be removed.
Region Purity (δ_{min})	This is the minimum purity a CH region must meet with respect to the target class (default = 0.8). Failure to meet this threshold causes the penalty to be applied in the GA objective function (Φ).
Partition Simplification Ratio (β)	This parameter specifies the threshold ratio of the min:max increase in error caused by removing a term from a partition. If the actual ratio is less than this threshold, then the term corresponding to the minimum increase in error can be removed from the partition. The choices vary between 0.0 and 1.0, with a default of 0.2.
Partition Simplification Error threshold (α)	This parameter specifies the maximum increase in error resulting from the elimination of a term from a partition. If the minimum increase in error does not exceed this threshold, then the corresponding term can be removed from the partition. The choices vary between 0.0 and 1.0, with a default of 0.5.
Partition Usage:	This parameter selects the type of partitions used during the CH region search. The choices are: mixed (global and local) or global only or local only.
Simplify Partition On:	This parameter selects the type of data set used as the basis for partition simplification. The choices are: full training set, winnowed training set, boundary point set, and weighted boundary point set (default).

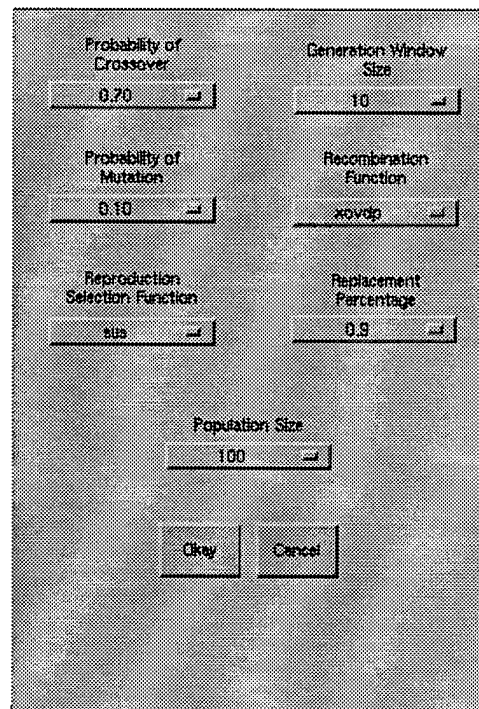


Figure 105 GRaCCE - Genetic Algorithm Menu.

Partition Utilization	Sample Size for Covariance Estimate
1.0	Automatic
Region Utility Ratio	Degree of Class Purity for Regions
0.01	0.80
Partition Simplification Ratio	Partition Simplification Error Threshold
0.20	0.50
Partition Usage:	
Mixed	
Simplify Partitions On:	
Weighted Boundary Points	
OK	Cancel

Figure 106 GRaCCE - Region Identification Phase Menu.

Appendix C. Sample GRaCCE Report

Table 30 Explanations of Fields within the GRaCCE Execution Report.

Field	Description
Dataset	Name of processed data set and training information.
GA Parameters	Utilized values of the genetic algorithm parameters described in Table 28.
Preprocessing Parameters	Summary of the utilized parameter values for the Region Identification phase as described in Table 29.
Feature Selection Status	Specifies the features utilized (with respect to the original data set).
Class Tallies	Provides the per class tallies of instances in the training and test data sets.
Clustering Level Attained	Lists the percentage of data (per class) enclosed in a CH region during training.
Classification Error Rates	Provides the classification error rates for the training and test data sets. A comprehensive explanation of these categories can be found in Section 6.4.
Partition Usage Metrics	The following metrics are provided: total partitions utilized in the final decision rule set; average number of partitions per CH region (cluster) and average decrease in the dimensionality of each partition (when the Partition Simplification phase is enabled). Dimensionality is defined as the number of non-zero terms in a given partition's vector.
Partition List	This is the mathematical description of each partition. Since a partition (h) is defined by Equation 15, then the components \vec{v}_N and X_0 are provided here for each partition.
Cluster List	Lists each CH region found during training. The CH regions are described in terms of class, number of members, position of the region's center (X_c) and number of defining partitions. A mapping is also provided to the partitions (in the above list) which enclose the CH region. This includes the orientation ($f_o = \pm 1$) of the partition with respect to X_c such that: $f_o h(X_c) \geq 0$. <i>A comprehension of this relationship is essential to interpreting the decision rule set.</i>

GPCEE Execution Report:

Dataset: wine23 (fold 3 of 5)

GA Parameters ...
Population Size ==> 100
Probability - Crossover ==> 0.70
Probability - Mutation ==> 0.10
Probability - Replacement ==> 0.9
Selection Function ==> sus
Recombination Function ==> xovdp
GA Window Size (Gen) ==> 10

Preprocessing Parameters ...
kNN (value of k) ==> 3
Assignment Halt Threshold ==>
Required Cluster Purity ==> 0.80
Minimum Boundaries ==> 1.0
Boundary Increment ==> 0.15
Partition Error Threshold ==> 0.50
Partition Min/Max Ratio ==> 0.10
Region Utility Ratio ==> 0.01
Simplification Done Using ==> Weighted Boundary Points

Feature Selection Parameters ...
Feature Selection Status ==> FULL

Training Data Information			...
Count for Class	0	==>	49
Count for Class	1	==>	55
Count for Class	2	==>	37
Total		==>	141

Test Data Information			...
Count for Class	0	==>	10
Count for Class	1	==>	16
Count for Class	2	==>	11
Total		==>	37

Clustering Level Attained			...
Level for Class	0	==>	1.000
Level for Class	1	==>	1.000
Level for Class	2	==>	1.000

Train Error Rate (Method 1)			...
Error Rate for Class	0	==>	0.000
Error Rate for Class	1	==>	0.000
Error Rate for Class	2	==>	0.000
Composite		==>	0.000

Train Error Rate (Method 2)			...
Error Rate for Class	0	==>	0.000
Error Rate for Class	1	==>	0.000
Error Rate for Class	2	==>	0.000

Composite ==> 0.000

Eval Error Rate (Method 1) ...

Error Rate for Class 0 ==> 0.000

Error Rate for Class 1 ==> 0.073

Error Rate for Class 2 ==> 0.000

Composite ==> 0.028

Eval Error Rate (Method 2) ...

Error Rate for Class 0 ==> 0.000

Error Rate for Class 1 ==> 0.018

Error Rate for Class 2 ==> 0.000

Composite ==> 0.007

Eval Error Rate (Method 3) ...

Error Rate for Class 0 ==> 0.000

Error Rate for Class 1 ==> 0.018

Error Rate for Class 2 ==> 0.000

Composite ==> 0.007

Test Error Rate (Method 1) ...

Error Rate for Class 0 ==> 0.000

Error Rate for Class 1 ==> 0.188

Error Rate for Class 2 ==> 0.000

Composite ==> 0.081

Test Error Rate (Method 2) ...

Error Rate for Class 0 ==> 0.000

Error Rate for Class 1 ==> 0.125
Error Rate for Class 2 ==> 0.000
Composite ==> 0.054

Test Error Rate (Method 3) ...

Error Rate for Class 0 ==> 0.000
Error Rate for Class 1 ==> 0.125
Error Rate for Class 2 ==> 0.000
Composite ==> 0.054

Eval Error Rate (Method 4) ...

Error Rate for Class 0 ==> 0.000
Error Rate for Class 1 ==> 0.073
Error Rate for Class 2 ==> 0.027
Composite ==> 0.035

Eval Error Rate (Method 5) ...

Error Rate for Class 0 ==> 0.000
Error Rate for Class 1 ==> 0.036
Error Rate for Class 2 ==> 0.027
Composite ==> 0.021

Test Error Rate (Method 4) ...

Error Rate for Class 0 ==> 0.000
Error Rate for Class 1 ==> 0.125
Error Rate for Class 2 ==> 0.000
Composite ==> 0.054

Test Error Rate (Method 5) ...

Error Rate for Class	0	==> 0.000
Error Rate for Class	1	==> 0.125
Error Rate for Class	2	==> 0.000
Composite		==> 0.054

Total Partitions Used	==> 3
Avg Partitions per Cluster	==> 1.667
Avg Decrease in Partition Size	==> 0.821

Partition 1 ...

V(1)	==> 0.421
V(2)	==> 0.000
V(3)	==> 0.730
V(4)	==> 0.000
V(5)	==> 0.000
V(6)	==> 0.000
V(7)	==> 0.000
V(8)	==> 0.000
V(9)	==> 0.000
V(10)	==> 0.000
V(11)	==> 0.000
V(12)	==> 0.000
V(13)	==> 0.002
X(1)	==> 12.765
X(2)	==> 1.962
X(3)	==> 2.324

X(4)	==> 19.013
X(5)	==> 97.249
X(6)	==> 2.479
X(7)	==> 2.393
X(8)	==> 0.341
X(9)	==> 1.695
X(10)	==> 3.933
X(11)	==> 1.062
X(12)	==> 2.899
X(13)	==> 730.379

Partition 2 ...

V(1)	==> 0.000
V(2)	==> 0.000
V(3)	==> 0.000
V(4)	==> 0.000
V(5)	==> 0.000
V(6)	==> 0.000
V(7)	==> 0.000
V(8)	==> 0.000
V(9)	==> 0.000
V(10)	==> 0.000
V(11)	==> 0.000
V(12)	==> 0.263
V(13)	==> 0.001
X(1)	==> 13.513
X(2)	==> 2.706
X(3)	==> 2.450
X(4)	==> 19.559

X(5)	==> 103.072
X(6)	==> 2.182
X(7)	==> 1.759
X(8)	==> 0.372
X(9)	==> 1.510
X(10)	==> 6.883
X(11)	==> 0.850
X(12)	==> 2.321
X(13)	==> 862.193
Partition 3	...
V(1)	==> 0.000
V(2)	==> 0.000
V(3)	==> 0.000
V(4)	==> 0.000
V(5)	==> 0.000
V(6)	==> 0.000
V(7)	==> 0.779
V(8)	==> 0.000
V(9)	==> 0.000
V(10)	==> -0.154
V(11)	==> 0.000
V(12)	==> 0.000
V(13)	==> 0.000
X(1)	==> 12.788
X(2)	==> 2.697
X(3)	==> 2.357
X(4)	==> 20.782
X(5)	==> 96.509

X(6)	==> 1.956
X(7)	==> 1.401
X(8)	==> 0.404
X(9)	==> 1.372
X(10)	==> 5.377
X(11)	==> 0.866
X(12)	==> 2.198
X(13)	==> 579.204

Total Clusters Found	==> 3
----------------------	-------

Cluster Number	==> 1
Primary Class	==> 0
Number of Members	==> 49
Number of Partitions	==> 2
Partition Map	...
Partition 1	==> 1
Partition 2	==> 1

Cluster Center	...
X(1)	==> 13.750
X(2)	==> 1.972
X(3)	==> 2.459
X(4)	==> 16.965
X(5)	==> 106.327
X(6)	==> 2.841
X(7)	==> 2.988
X(8)	==> 0.291
X(9)	==> 1.896

X(10)	==> 5.518
X(11)	==> 1.064
X(12)	==> 3.131
X(13)	==> 1132.286
Cluster Number	==> 2
Primary Class	==> 1
Number of Members	==> 55
Number of Partitions	==> 2
Partition Map	...
Partition 1	==> -1
Partition 3	==> 1
Cluster Center	...
X(1)	==> 12.255
X(2)	==> 1.957
X(3)	==> 2.254
X(4)	==> 20.075
X(5)	==> 92.545
X(6)	==> 2.291
X(7)	==> 2.085
X(8)	==> 0.367
X(9)	==> 1.591
X(10)	==> 3.112
X(11)	==> 1.060
X(12)	==> 2.778
X(13)	==> 522.145
Cluster Number	==> 3

Primary Class	==> 2
Number of Members	==> 37
Number of Partitions	==> 1
Partition Map	...
Partition 3	==> -1
Cluster Center	...
X(1)	==> 13.257
X(2)	==> 3.348
X(3)	==> 2.447
X(4)	==> 21.405
X(5)	==> 100.000
X(6)	==> 1.661
X(7)	==> 0.799
X(8)	==> 0.437
X(9)	==> 1.180
X(10)	==> 7.372
X(11)	==> 0.694
X(12)	==> 1.687
X(13)	==> 629.459

Appendix D. Summary of Mathematical and Algorithmic Notation

Table 31 General Notation Summary.

Symbol	Description
d	Number of features (dimensionality) in a given data set.
m	Number of classes in a given data set.
n	Number of instances (feature vectors) in a given data set.
Ω	Data set.
ω_i	Subset of data set belonging to the i th class.
D	Euclidean distance function between two points.
D_M	Mahalanobis distance function between two points.

Table 32 Notation Summary - Boundary Points.

b	Represents an individual boundary point (refer to Definition 1).
Λ	Set of boundary points generated from a given data set Also see description of b (above).
λ_i	Subset of boundary points belonging to the i th class.
ρ	Function that returns weight for a given boundary point (refer to Definition 3).

Table 33 Notation Summary - Class Partition.

\vec{v}_i	The vector normal to the i th class partition.
h	Denotes a hyper-plane (partition) separating two classes (refer to Equation 15).
X_0	Anchor point of partition (refer to Equation 13).
Γ	Set of inter-class partitions generated from a given data set. Also see description of h (above).
γ_i	Subset of partitions associated with the i th class. Note that since each partition separates two classes, any given partition is associated with two distinct classes.
μ	Statistic - mean.
σ	Statistic - standard deviation.
Σ	Covariance matrix.

Table 34 Notation Summary - Genetic Algorithm (GA).

\vec{a}_i	The i th individual (chromosome) within the GA population.
P	Represents the population of a GA.
p	Size of the GA population.
q	Size of the GA convergence window (in generations).
I	Template for GA chromosome structure.
l	Length of GA chromosome.
f_m	Mapping function between genes in I and associated partitions.
f_o	Orientation function (returns ± 1) for partition associated with each gene in I .
Φ	Genetic algorithm objective function.
ϕ_i	This is the i th component of the objective function.

Table 35 Notation Summary - Region Identification (RI) Phase.

t	Designator of target class for CH region search.
\vec{z}	Deterministic solution found using pseudocode in Figure 33.
δ	Returns the error (purity) of a the region defined by a given GA chromosome with respect to the target class.
δ_{min}	Minimum acceptable error within a given CH region.
R	Set of class homogeneous (CH) region generated by GRaCCE.
\vec{r}_i	The i th region in R . This is an array of settings ($-1/0/1$) for each partition.

Table 36 Notation Summary - Region Refinement (RR) Phase.

RUR	Function which computes the region utility ratio fro a given region (refer to Equation 19).
RUR_{min}	The minimum acceppable region utility ratio. Regions which a lower value than this threshold are eliminated.
κ_t	Function which computes the proportion of class ω_t enclosed by a given region.

Table 37 Notation Summary - Partition Simplification Algorithm.

α	Minimum allowable increase in error for for simplifying a given partition.
β	Minimum ration of minimum to maximum increase in error for simplifying a given partition.
ϵ	Error rate (for the data set).
$(\epsilon)_{min}^+$	Mininum increase in ϵ resulting from a change to a partition in Γ .
$(\epsilon)_{max}^+$	Maximum increase in ϵ resulting from a change to a partition in Γ .
j_{min}	Index of partition term that results in $(\epsilon)_{min}^+$ when set to 0.

Appendix E. Description of cGRaCCE Hardware Configuration

The target platform for cGRaCCE is a cluster of personal computers (PCs), known as the AFIT Beowulf Cluster. This heterogeneous PC cluster consists of one Dell 450 MHz, six Dell 400 MHz, and four Gateway 333 MHz single-processor Pentium II computers connected via a 100 Mbps full duplex fast Ethernet switch. The I/O bus on the Gateways operates at 66 MHz; the Dells I/O bus is clocked at 100 MHz. All of the cGRaCCE experiments are run using Windows NT on the Dell 400 MHz processors. Each of the Dell processors has 128 MB of 10 nsec SDRAM and one 8.4 GB SCSI hard drive. The MPI/Pro 1.2.3 communication package is utilized to handle interprocessor communications. A block diagram of the Beowulf configuration is provided in Figure 107.

Vita

Major Robert E. Marmelstein was born on 20 July 1963 in New York City, New York. He graduated *cum laude* from Michigan Technological University in May 1985 with a Bachelor of Science degree in Electrical Engineering, and accepted a commission as a Second Lieutenant in the United States Air Force.

Major Marmelstein's first assignment was to the USAF Avionics Laboratory. While there he managed programs to mature the Ada programming language, such as the Ada Compiler Evaluation Capability (ACEC). He transferred in September 1989 to the Electronic Systems Center (ESC) where he was software manager for the 316F, Cobra Dane System Modernization (CDSM) and Joint Service Image Processing System (JSIPS) programs. In December 1990, he graduated from the University of Massachusetts at Lowell with a Master of Science in Computer Engineering. His next assignment (October 1993) was to the 513th Engineering and Test Squadron (ETS). While there, he served as chief of the B-1 defensive systems engineering branch, which programmed the ALQ-161A defensive avionics system for all B-1 bomber operational and test missions. In June 1996, he was assigned to AFIT to begin his PhD studies.

Major Marmelstein's professional achievements include an Air Force invention and a US Patent (#5,187,788) for developing an automatic code generation tool for Ada. In addition, he received a monetary USAF suggestion award for developing an emitter ambiguity analysis system for the ALQ-161A.

In addition to his military experience, Major Marmelstein worked as a test engineer for the Chicago Transit Authority where he maintained subway turnstile and bus farebox software. He also wrote a book titled *Programming Computer Games with C*, which was published by M&T Press in 1994. Lastly, he taught undergraduate courses in assembly language and C++ at Bellevue University and the University of Nebraska at Omaha.

Permanent address: 3012 Homeworth Lane
Beavercreek, OH 45434

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 4, 1999	3. REPORT TYPE AND DATES COVERED Final Report (July, 1996 - June, 1999)	
4. TITLE AND SUBTITLE Evolving Compact Decision Rule Sets			5. FUNDING NUMBERS	
6. AUTHOR(S) Robert E. Marmelstein				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Wright-Patterson Air Force Base 2950 P Street WPAFB, OH 45433-7765			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratories (AFRL)/IFTA WPAFB, OH 45433			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION AVAILABILITY STATEMENT Distribution unlimited. Available for public release.			12b. DISTRIBUTION CODE DOD	
13. ABSTRACT (Maximum 200 words) While data mining technology holds the promise of automatically extracting useful patterns (such as decision rules) from data, this potential has yet to be realized. One of the major technical impediments is that the current generation of data mining tools produce decision rule sets that are very accurate, but extremely complex and difficult to interpret. As a result, there is a clear need for methods that yield decision rule sets that are both accurate and compact. The development of the Genetic Rule and Classifier Construction Environment (GRaCCE) is proposed as an alternative to existing decision rule induction (DRI) algorithms. GRaCCE is a multi-phase algorithm which harnesses the power of evolutionary search to mine classification rules from data. These rules are based on piece-wise linear estimates of the Bayes decision boundary within a winnowed subset of the data. Once a sufficient set of these hyper-planes are generated, a genetic algorithm (GA) based 0/1 search is performed to locate combinations of them that enclose class homogeneous regions of the data. It is shown that this approach enables GRaCCE to produce rule sets significantly more compact than those of other DRI methods while achieving a comparable level of accuracy. Since the principle of Occam's razor tells us to always prefer the simplest model that fits the data, the rules found by GRaCCE are of greater utility than those identified by existing methods.				
14. SUBJECT TERMS Data Mining, Machine Learning, Genetic Algorithms, Decision Trees, Pattern Recognition.			15. NUMBER OF PAGES 273	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unl	